

DECLARATION OF NATHANIEL E FRANK-WHITE

1. I am a Records Request Processor at the Internet Archive. I make this declaration of my own personal knowledge.
2. The Internet Archive is a website that provides access to a digital library of Internet sites and other cultural artifacts in digital form. Like a paper library, we provide free access to researchers, historians, scholars, and the general public. The Internet Archive has partnered with and receives support from various institutions, including the Library of Congress.
3. The Internet Archive has created a service known as the Wayback Machine. The Wayback Machine makes it possible to browse more than 450 billion pages stored in the Internet Archive's web archive. Visitors to the Wayback Machine can search archives by URL (i.e., a website address). If archived records for a URL are available, the visitor will be presented with a display of available dates. The visitor may select one of those dates, and begin browsing an archived version of the Web. Links on archived files in the Wayback Machine point to other archived files (whether HTML pages or other file types), if any are found for the URL indicated by a given link. For instance, the Wayback Machine is designed such that when a visitor clicks on a hyperlink on an archived page that points to another URL, the visitor will be served the archived file found for the hyperlink's URL with the closest available date to the initial file containing the hyperlink.
4. The archived data made viewable and browseable by the Wayback Machine is obtained by use of web archiving software that automatically stores copies of files available via the Internet, each file preserved as it existed at a particular point in time.
5. The Internet Archive assigns a URL on its site to the archived files in the format `http://web.archive.org/web/[Year in yyyy][Month in mm][Day in dd][Time code in hh:mm:ss]/[Archived URL]` aka an "extended URL". Thus, the extended URL `http://web.archive.org/web/19970126045828/http://www.archive.org/` would be the URL for the record of the Internet Archive home page HTML file (`http://www.archive.org/`) archived on January 26, 1997 at 4:58 a.m. and 28 seconds (1997/01/26 at 04:58:28). The date indicated by an extended URL applies to a preserved instance of a file for a given URL, but not necessarily to any other files linked therein. Thus, in the case of a page constituted by a primary HTML file and other separate files (e.g., files with images, audio, multimedia, design elements, or other embedded content) linked within that primary HTML file, the primary HTML file and the other files will each have their own respective extended URLs and may not have been archived on the same dates.
6. Attached hereto as Exhibit A are true and accurate copies of screenshots of the Internet Archive's records of the archived files for the URLs and the dates specified in the attached coversheet of each printout.

7. Attached hereto as Exhibit B are true and accurate copies of the Internet Archive's records of the archived files for the URLs and the dates specified in the attached coversheet of each file.
8. I declare under penalty of perjury that the foregoing is true and correct.

DATE: December 12, 2022

Nathaniel E Frank-White
Nathaniel E Frank-White

EXHIBIT A

Wayback Machine

147 captures

24 Feb 1998 - 5 Oct 2022

Go

AUGOCTDEC

09

200220032004

About this capture

United States Home | Select a Location

Site Map | Contact Us | About Intel

Products | Support

Home Computing | Business Computing | Developer | Reseller / Solutions

Advanced Search

Developer

Hardware Design

Software Development

Download Software/Drivers

Design Support/Services

Events, Training & Publications

R&D Initiatives

server home

Intel® Server Products

Server Leadership Technologies

Intel® Smart Kit Tool

Server Summit

Resellers & Dealers

Community

Industry Technologies and Specifications

Intel® Solution Services

Intel® Technical Documentation Center

documentation

IPMI Home

What is IPMI?

Specifications

Downloads

License Agreements

Industry Adopters

Support

SAF-TE Home

Intelligent Platform Management Interface



New revision available (updated on 10/1/03):

- IPMI v2.0 DRAFT specifications for Adopters Review**
This document presents proposals for new "Serial Over LAN" and "RMCP+" (Remote Management Control Protocol+) extensions to the IPMI v1.5 specifications.
- Addenda, Errata, and Clarifications document revision 4 for IPMI v1.5 rev 1.1 specification**
This revision lists the cumulative addenda, errata, and clarifications against the rev. 1.1 release of the IPMI v1.5 specification. This errata adds two items, E268 and E291, that were missing in the errata 3 document but were present in the associated 9/12/03 markup.
- IPMI v1.5 rev 1.1 specification markup for errata revision 4**
This is a version of the IPMI v1.5 rev 1.1 specifications with revisions marked per the Addenda, Errata, and Clarifications document revision 4. This replaces the 9/12/03 markup that accompanied errata 3. The previous markup missed including the optional 5th byte for the Get Chassis Status command that was part of errata E317.

Recent Updates (updated on 5/02/03):

- IPMI v1.5 Conformance Test Suite (ICTS) Prototype 5.05** 4/15/03
Includes IPMI v1.0 and IPMI v1.5 automated conformance tests, IPMI v1.5 CMDTOOL for manual IPMI v1.5 testing, support for PCI* card based IPMB and SMBus testing, and support for IPMI v1.5 new interfaces including LAN, Serial and SMBus. ICTS 5.05 is the latest update and adds new functional and interface protocol tests for IPMI 1.5 and includes some bug fixes as well.
- ICMB Specification V1.0 Document Revision 1.3** 4/2/03
ICMB v1.0 Revision 1.3 adds new commands for supporting Group Chassis Control capabilities. Group Chassis Control provides a mechanism to enable power, reset, and diagnostic interrupt control of multiple chassis using a single broadcast command on the ICMB.
- IPMI v1.5 Reference Driver for IA-64 and IA-32 under Windows* .NET/2000 OS**
IPMI 1.5 reference driver implementation for IA-64 and IA-32 under Windows* .NET/2000 OS available to IPMI adopters only.
- IPMI v1.5 Revision 1.1** 2/20/02
Intel Corporation, Hewlett Packard Company, NEC Corporation, and Dell Computer Corporation are pleased to announce the availability of revision 1.1 of the Intelligent Platform Management Interface (IPMI) v1.5 specifications. This update of the IPMI v1.5 specification incorporates important errata and addenda that address issues discovered in the original revision 1.0 release of the specification. A companion addenda and errata document (revision 5, below) separately lists the errata and addenda that have been included.

To receive notification of website updates and upcoming industry events, please send us your email address and name using the [feedback form](#).

Intel Corporation, Hewlett Packard Company, NEC Corporation, and Dell Computer Corporation are pleased to announce the availability of version 1.5 of the Intelligent Platform Management Interface (IPMI) specifications. IPMI defines common interfaces to "Intelligent" hardware used to monitor a server's physical health characteristics, such as temperature, voltage, fans, power supplies and chassis. These capabilities provide information that enables system management, recovery, and asset tracking which help drive down the total cost of ownership (TCO). The new interfaces in IPMI v1.5 facilitate the management of rack-mounted Internet servers and systems in remote environment over serial, modem and LAN connections. A superset of version 1.0, IPMI v1.5 also includes new capabilities to automatically alert IT managers of system errors and enable immediate system recovery. These new capabilities combined with the remote management functionality allow IT managers to manage their servers and systems, regardless of system health, power state or supported communication media. The [specifications](#) interoperability guidelines will enable server OEMs to quickly bring new server hardware with these advanced capabilities to market efficiently and cost effectively. See [IPMI v1.5](#) and [IPMI v1.0](#) press announcements for additional information.

Contributors Agreement:

Before contributing feedback to the IPMI specifications, a Contributors Agreement must be signed. All feedback will be carefully tracked, evaluated and incorporated as appropriate however there is no guarantee that feedback will be included in the future releases of the specifications. Please follow the steps below:

- Download and print the [Contributors Agreement](#).
- Sign the Contributors Agreement
- Fax Contributors Agreement to (please include your email address):
Intel Corporation
Attn: IPMI Initiative
Fax: (503) 712-1428
- A website address will be sent to you via email immediately upon receiving your signed Contributors Agreement. From this website you will be able to provide your feedback.

Adopters Agreement:

Before implementing the IPMI, IPMB or ICMB specifications, a royalty-free reciprocal patent license must be signed. Please follow the steps below to sign the IPMI Adopters Agreement:

- Download and print 2 copies of the [Adopters Agreement](#).
- Sign both copies and send to:
Intel Corporation
JFS-301
5200 N.E. Elam Young Parkway
Hillsboro, OR 97124-6497
Attn: Fadi Zuhayri
- Include mailing address. One copy will be sent back for your records.

For general questions and comments submit your feedback [here](#).

THESE SPECIFICATIONS ARE PROVIDED "AS IS" WITH NO WARRANTIES, and subject to change without notice.

[back to top](#)

*Legal Information | Privacy Policy

©2003 Intel Corporation

EXHIBIT B

https://web.archive.org/web/20030610155215/http://support.intel.com/idf/us/spr2003/presentations/S03USOSAS41_OS.pdf

Advances in Intelligent Platform Management

**Tom Slaight
Server Management Architect
Enterprise Platforms Group
Intel Corporation**



Agenda

- IPMI Update
- New system directions for IPMI
- New components for IPMI
- IPMI future directions
- Summary



Itanium® is a trademark or registered trademark of Intel Corporation or its subsidiaries in the United States or other countries."



Lenovo

Ex. 1039 - Page 9

Introduction

- **Audience:**
Architects, Technical Managers, Firmware Leads, and Hardware Designers
 - Involved in architecture, component selection, debug, test, or design of server baseboard and peripheral management subsystems
- **Focus: IPMI-based implementations**
 - Hardware and software components

IPMI

Intelligent Platform Management Interface

- Defines a standardized, abstracted, message-based interface to intelligent platform management hardware
- Defines standardized records for describing platform management devices and their characteristics

Promoters:



Adopters: Over 138 and growing

<http://developer.intel.com/design/servers/ipmi>



Lenovo

Ex. 1039 - Page 11

Initiative News



The screenshot shows the Intel Developer website interface. At the top, there's a navigation bar with 'Products' and 'Support' buttons, and links for 'Home Computing', 'Business', 'Developer', and 'Reseller'. A search bar is on the right. The left sidebar has a 'Developer' section with links to 'Hardware Design', 'Software Development', 'Download Software/Drivers', 'Design Support/Services', 'Events, Training & Pubs', and 'R&D/Initiatives'. Below this is a 'server home' section with links to 'Server Building Blocks', 'Itanium™ Processor Family', 'Hardware Developer's Resource Center', 'Software Developer's Resource Center', 'Industry Technical Specifications', 'Community', 'Related Sites', and 'Tools & Resources'. The main content area features a large image of server hardware and a banner for 'Intelligent Platform Management Interface'. A large blue diagonal banner with white text reads: 'Updated Errata, ICMB specifications, and Conformance Test Suite'. Below this banner, the text 'New releases (updated on 8/05/02):' is followed by two links: 'Reference Driver for IA-64 and IA-32 under Windows* .NET/2000 OS' and 'IPMI v1.5 Conformance Test Suite (ICTS) Prototype 5.02'. The second link is followed by a description: 'Includes IPMI v1.0 and IPMI v1.5 automated conformance tests, IPMI v1.5 CMDTOOL for manual IPMI v1.5 testing, support for PCI* card based IPMB and SMBus testing, and support for IPMI v1.5 new interfaces including LAN, Serial and SMBus. ICTS 5.02 is an update to ICTS 5.01 and adds new tests for IPMI 1.5 commands and includes some bug fixes as well.'

Updated Errata, ICMB specifications, and Conformance Test Suite

New releases (updated on 8/05/02):

- [Reference Driver for IA-64 and IA-32 under Windows* .NET/2000 OS](#)
Reference driver implementation for IA-64 and IA-32 under Windows* .NET/2000 OS available to IPMI adopters only.
- [IPMI v1.5 Conformance Test Suite \(ICTS\) Prototype 5.02](#)
Includes IPMI v1.0 and IPMI v1.5 automated conformance tests, IPMI v1.5 CMDTOOL for manual IPMI v1.5 testing, support for PCI* card based IPMB and SMBus testing, and support for IPMI v1.5 new interfaces including LAN, Serial and SMBus. ICTS 5.02 is an update to ICTS 5.01 and adds new tests for IPMI 1.5 commands and includes some bug fixes as well.

developer.intel.com/design/servers/ipmi

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



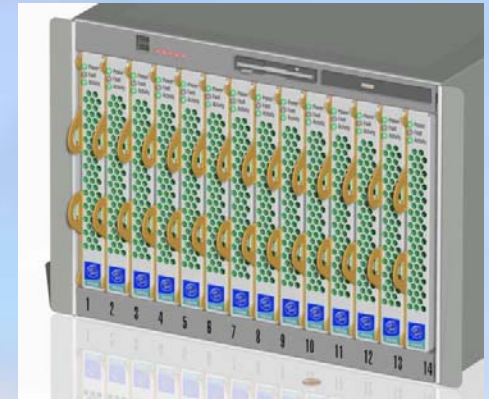
Agenda

- IPMI Update
- **New system directions for IPMI**
- New components for IPMI
- IPMI future directions
- Summary



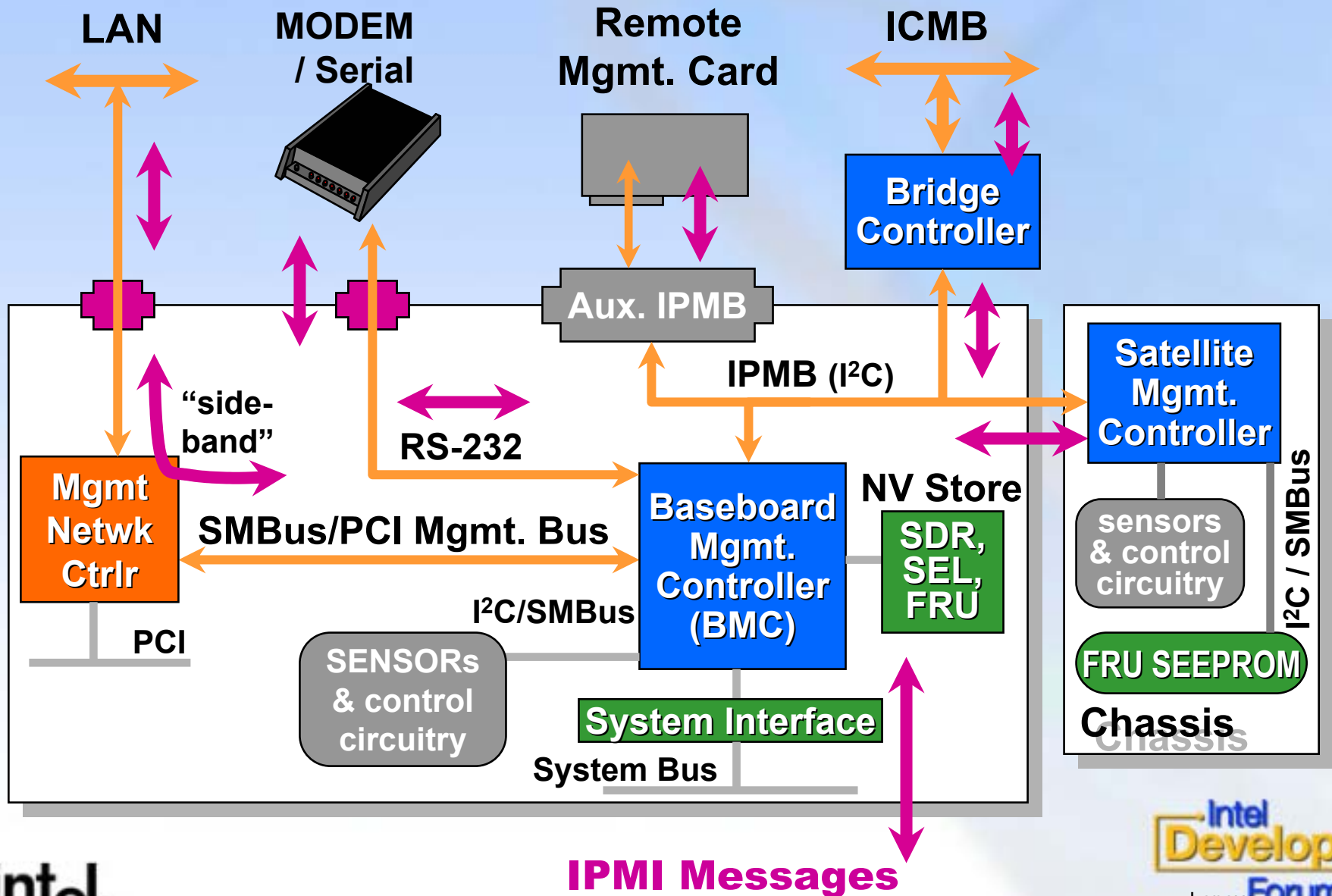
New System Directions for IPMI

- **Modular Systems**
 - Advanced TCA & General Modular
- **Low Cost Systems**
 - Low cost BMCs
 - Add-in BMCs
- **Group Managed Systems**
 - IPMI for Group Chassis Control

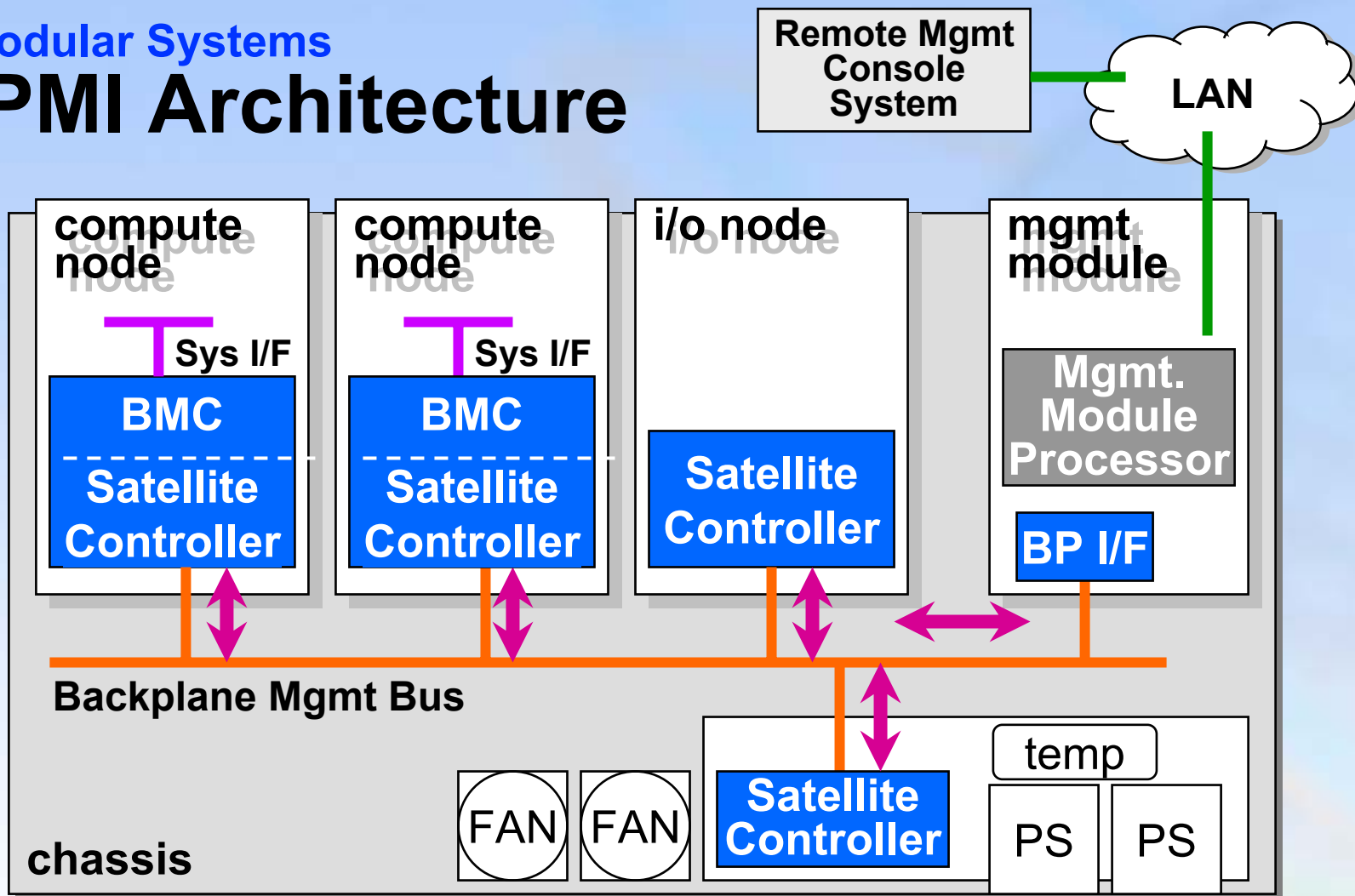


IPMI grows to enable competitive features across server classes

IPMI v1.5 Architecture



IPMI Architecture

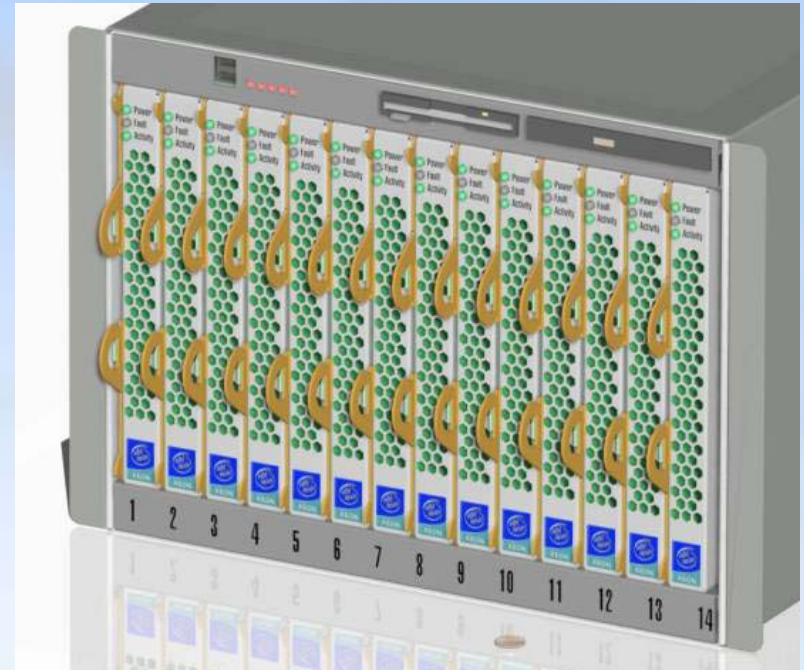


IPMI Messages ↔

More info in OSAS135: "Using IPMI Platform Management In Modular Computer Systems"

IPMI Spec Extensions

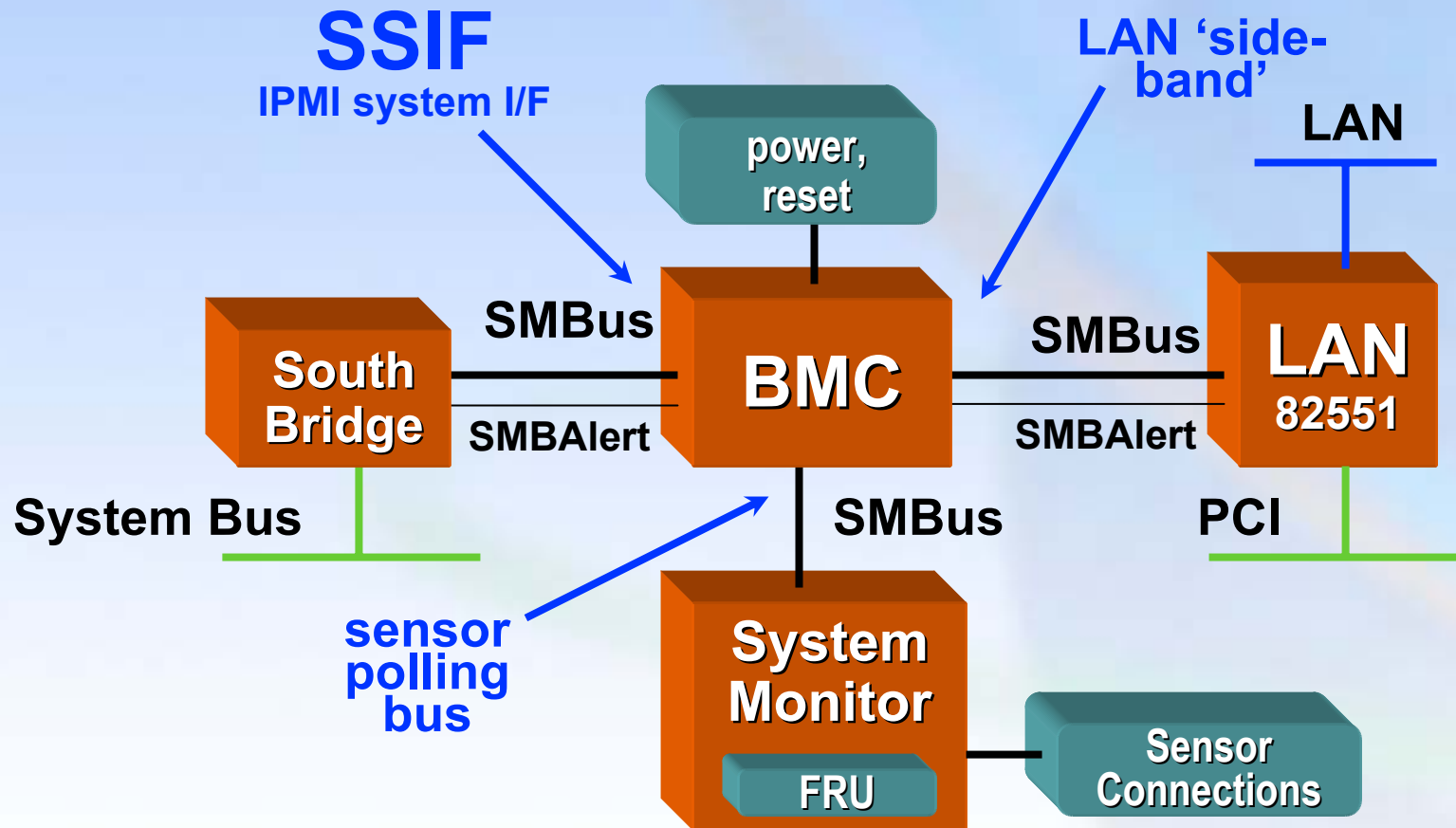
- Entity information extensions to identify node location in a given chassis
- Options to coordinate Node/FRU removal and replacement
- Management Bus 'failover' status
 - Supports redundant management bus connections



Baseline 'mini' BMCs

- IPMI Conformant 'Baseline' BMC
 - Covers mandatory IPMI BMC features
- 'Turnkey' firmware with limited customization and chassis support
 - "NVRAM" configurable to motherboard sensors
 - Minimized support for IPMI options
 - E.g. May be 'LAN only'
 - May have limited options such as # of Users
- Small, low pin-count packaging...

SSIF - SMBus System Interface



SSIF - SMBus System Interface

- **Provides BMC system interface access via SMBus**
 - Low pin count
 - Relies on controller-specific SMBus drivers
- **BMC accessed as SMBus Slave device**
 - Requests delivered using 'Block Write' protocol
 - Responses retrieved using 'Block Read' protocol
 - SMBAlert signal status change/message available
 - 'Get Status' command allows interface status to be polled
 - Includes ability to write/read more than 32 data bytes using two SMBus transactions
- **New 'Reserve Device' command**
 - Optional command directs BMC to suspend access to specified device for xx milliseconds
 - Resolves access to 'shared' SMBus devices

SSIF - Message Formats

Write_Message:

Sends encapsulated IPMI message data to controller

Format: Slave Addr | 0b, CMD=Write_Msg, Length,
Message Data, **CHECKSUM**

Read_Message:

Retrieves encapsulated IPMI message data from controller

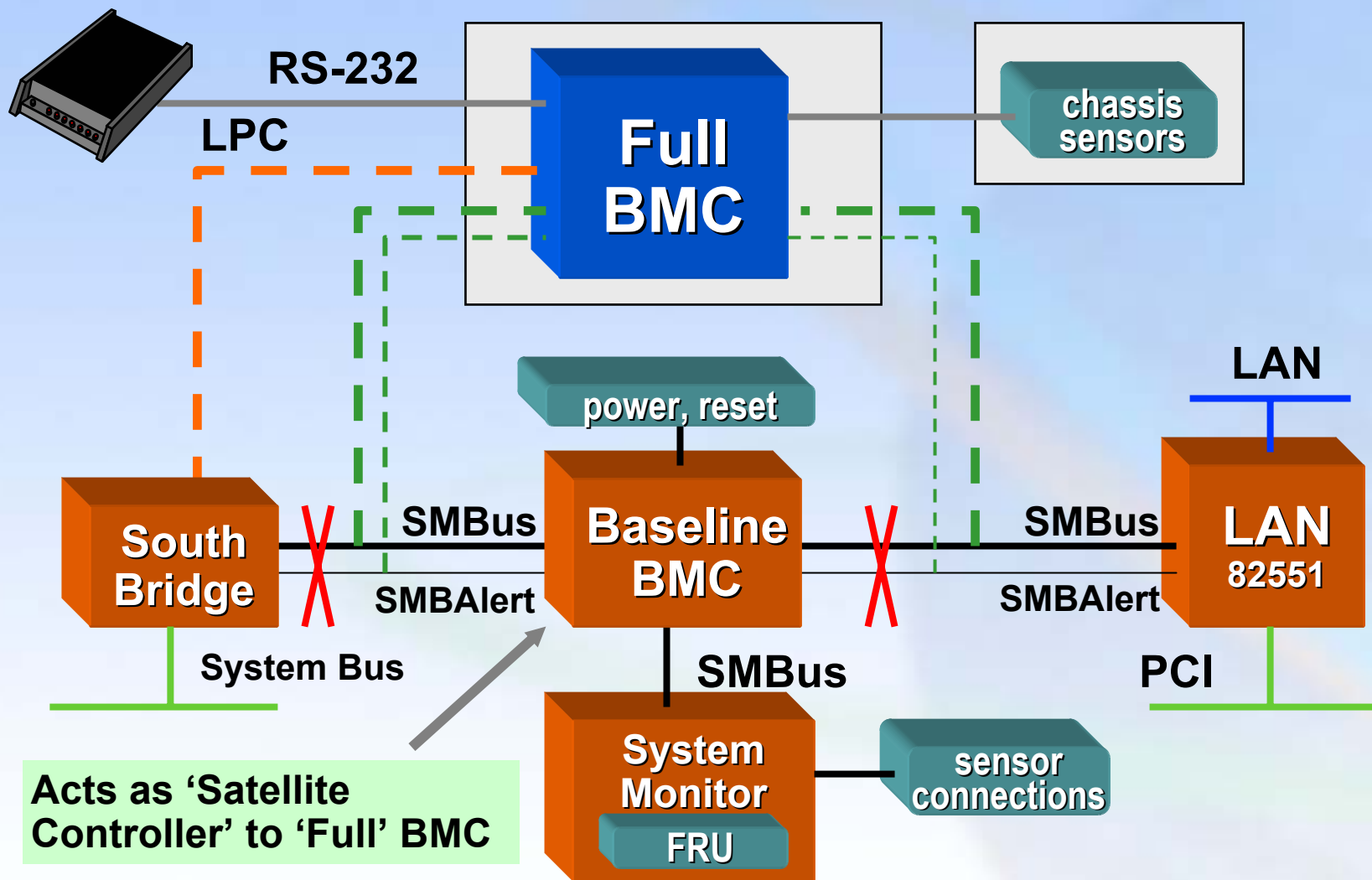
Format: Slave Addr | 0b, CMD=Read_Msg, Slave Addr | 1b,
Length, Message Data, **CHECKSUM**

Get_Status:

Indicates message data available for reading, or error/busy status
of interfaces

Format: Slave Addr | 0b, CMD=Get_Status, Slave Addr | 1b,
Length, Message Data, **CHECKSUM**

Multi-level BMC Options



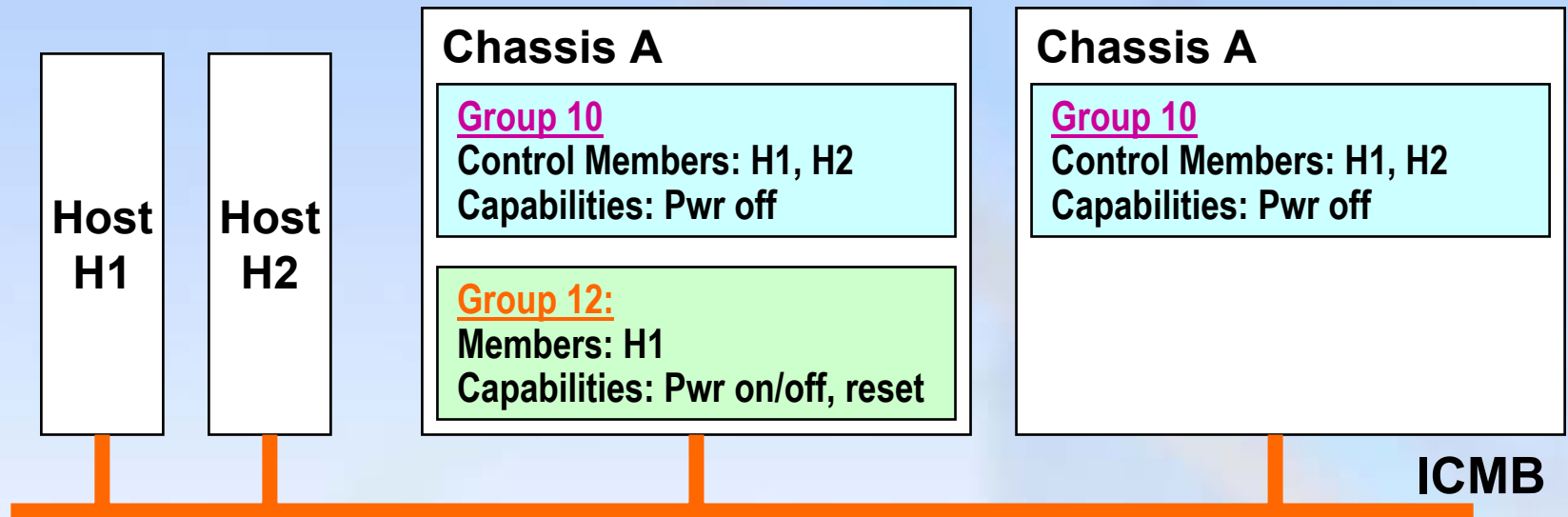
**Low Cost Options enable IPMI
for all Server classes**

ICMB Group Chassis Control

- **Chassis Control single broadcast**
 - power on/off, reset, diagnostic interrupt
- **Group-specific operation enables, e.g.**
 - “Group 1” enabled for power-on, -off, reset
 - “Group 2” enabled for power-on only
- **Enables Power/reset sequencing**
- **Enables ‘one button’ power on**
 - e.g. pressing power button on compute chassis automatically powers up associated peripheral chassis
- **Enables ‘service lockouts’**
 - e.g. powering down a chassis blocked until all ‘controlling members’ have requested same power state



ICMB Group Chassis Control



- Up to 255 different control groups per ICMB
- Each chassis can belong to up to 4 control groups
- A single 'Group Chassis Control' command can be targeted to 4 different control groups
- Each group has up to 16 'controlling members'
- Members can 'request' or 'force' the control state

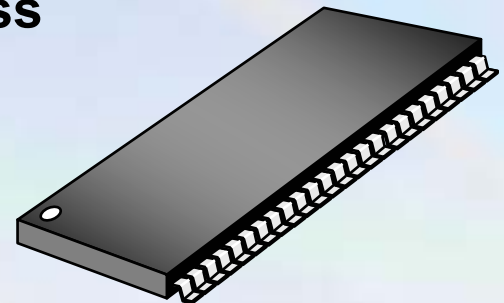
Agenda

- IPMI Update
- New system directions for IPMI
- **New components for IPMI**
- IPMI future directions
- Summary



National Semiconductor “Mini” BMC

- **PC87431M*** - Targets IPMI LAN remote mgmt.
 - supports monitoring by local mgmt. s/w via SMBus
 - configurable sensor polling
- **Internal FLASH, RAM, NVRAM**
 - up to 512 bytes NV available for OEM use
- **Authenticated IPMI LAN support for:**
 - System reset, SMI/NMI, and power control
 - settable ‘Boot Options’
 - FRU, System Event Log, and SDR access
 - Sensor access
 - Alerting via IPMI/PET SNMP Traps
- **Platform Event Filtering**
 - configurable actions on events
 - power control, reset, fault light, NMI/SMI, and alert



IPMI Software

IPMI Web Site

- **Reference drivers**
 - for Itanium™ Architecture and IA-32 under Windows* .NET/2000 and Linux
- **IPMI Conformance Test Suite**
 - Serves as both validation and development tool

developer.intel.com/design/servers/ipmi



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

IPMI Software

OSA Technologies

- Management Applications for IPMI
 - “Remote Console” applications and IPMI Drivers
- Firmware engineering also available
 - SDKs for popular BMCs
 - Supports IPMI v1.5 and out-of-band access (serial, LAN)
- IDF LAB Thursday

“Exploring Remote Manageability Building Blocks for Next Generation Servers and Devices”



**IPMI components improve TTM
and reduce design cost**

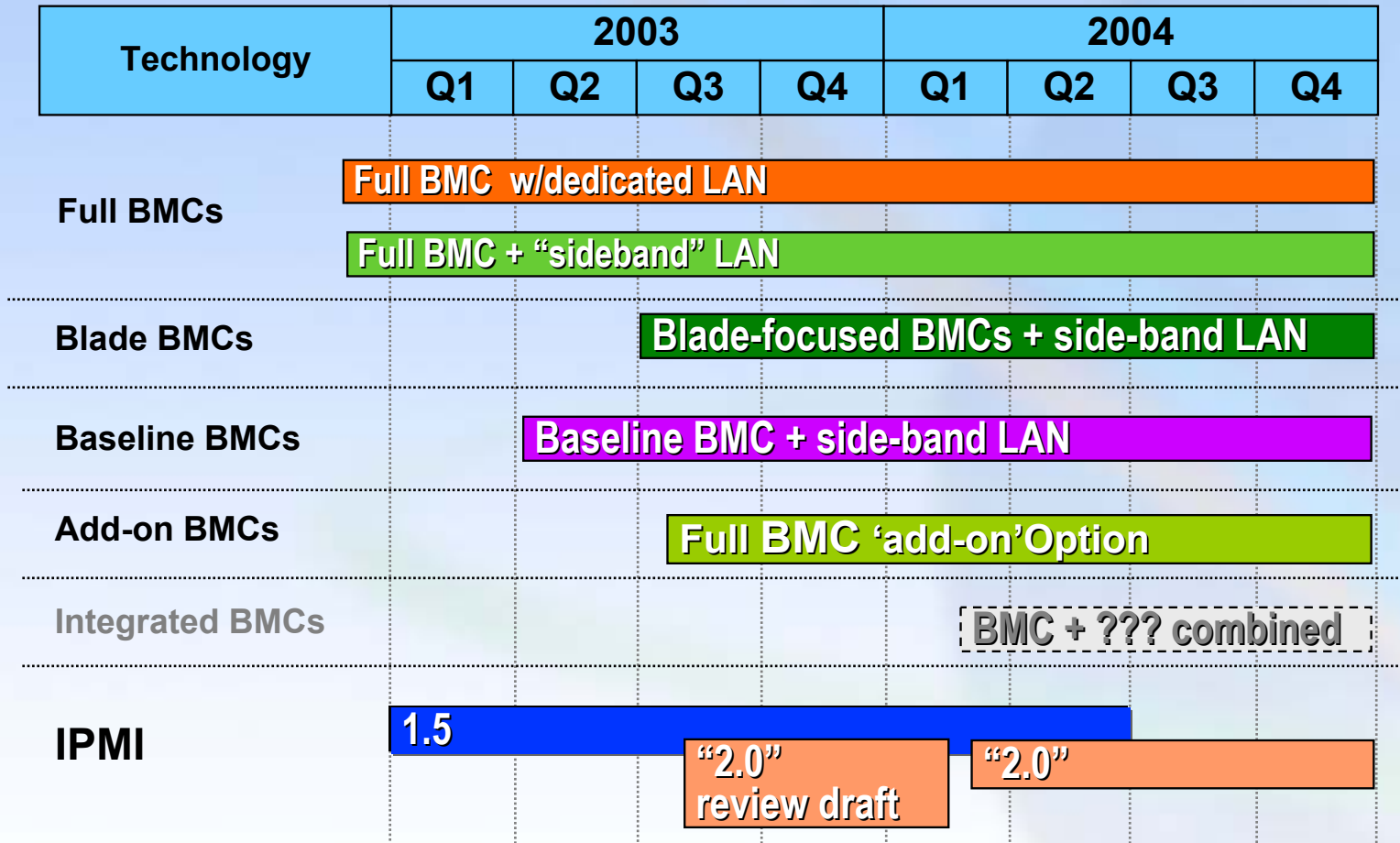
Agenda

- IPMI Update
- New system directions for IPMI
- New components for IPMI
- **IPMI future directions**
- Summary



IPMI Future Directions

Technology Transitions



Update next IDF

IPMI '2.0' Proposals

- **Serial redirection over LAN**
- **Terminal mode extensions (improved 'CLI')**
- **ASF Alignment**
 - Common authentication protocols
 - Smooths ASF to IPMI transition between desktop and sub-entry server systems
- **Modular (blade) support**
 - blade/chassis relationship, blade power mgmt., etc.
 - AdvancedTCA support (formerly 'CompactPCI')
- **IPMI over Web**
 - enabling technology for IPMI over Web (may be post 2.0)

**IPMI continues to evolve
valuable new capabilities**

Agenda

- IPMI Update
- New system directions for IPMI
- New components for IPMI
- IPMI future directions
- **Summary**



Summary

- **IPMI grows to enable competitive features across server classes**
- **Third party components improve TTM and reduce design cost**
- **IPMI continues to evolve valuable new capabilities**

developer.intel.com/design/servers/ipmi



Lenovo

Ex. 1039 - Page 33

Glossary

BMC	Baseboard Management Controller.
FRU	Field Replaceable Unit. A field replaceable component such as a board, module, fan, power supply, etc.
ICMB	Intelligent Chassis Management Bus. The ICMB provides a dedicated management bus that enables delivering IPMI messages and alerts between multiple host and peripheral chassis.
IPMB	Intelligent Platform Management Bus. Name for the architecture, protocol, and implementation of a special bus that interconnects the baseboard and chassis electronics and provides a communications media for system platform management information.
IPMI	Intelligent Platform Management Interface. IPMI defines a common, abstracted, and self-descriptive interface for platform management hardware that monitors server characteristics such as temperature, voltage, fans, power supplies, and chassis.
OOB	Out-of-Band. System platform management access that does not involve going through the OS or other software running on the main processors of the managed system.
PEF	Platform Event Filtering. A feature in IPMI that enables the BMC to generate a selectable action (e.g. power on/off, reset, send Alert, etc.) when a configurable event occurs on the management system.

Glossary

- SAF Service Availability Forum. Standards body consisting of Telco platform and software vendors that is defining RAS standards including UCMI
- SAF-HPI Server Availability Forum - Hardware Platform Interface. Name for a set of APIs and structures for representing and accessing platform management hardware.
- SAF-TE SCSI Accessed Fault-Tolerant Enclosures. SAF-TE provides a mechanism that enables RAID fault information to be sent to the hot-swap backplane via SCSI.
- SDR Sensor Data Record. SDRs provide the information that tells management software what sensors, events, management controllers, and FRU information is available from a given IPMI implementation.
- SEL System Event Log. A non-volatile storage area and associated interfaces for storing system platform event information for later retrieval.

IP Filter Based Firewalls HOWTO

Brendan Conoboy <synk@swcp.com>

Erik Fichtner <emf@obfuscation.org>

Mon Aug 28 23:41:36 EDT 2000

Abstract: This document is intended to introduce a new user to the IP Filter firewalling package and, at the same time, teach the user some basic fundamentals of good firewall design.

1. Introduction

IP Filter is a great little firewall package. It does just about everything other free firewalls (ipfwadm, ipchains, ipfw) do, but it's also portable and does neat stuff the others don't. This document is intended to make some cohesive sense of the sparse documentation presently available for ipfilter. Some prior familiarity with packet filtering will be useful, however too much familiarity may make this document a waste of your time. For greater understanding of firewalls, the authors recommend reading *Building Internet Firewalls*, Chapman & Zwicky, O'Reilly and Associates; and *TCP/IP Illustrated, Volume 1*, Stevens, Addison-Wesley.

1.1. Disclaimer

The authors of this document are not responsible for any damages incurred due to actions taken based on this document. This document is meant as an introduction to building a firewall based on IP-Filter. If you do not feel comfortable taking responsibility for your own actions, you should stop reading this document and hire a qualified security professional to install your firewall for you.

1.2. Copyright

Unless otherwise stated, HOWTO documents are copyrighted by their respective authors. HOWTO documents may be reproduced and distributed in whole or in part, in any medium physical or electronic, as long as this copyright notice is retained on all copies. Commercial redistribution is allowed and encouraged; however, the authors would like to be notified of any such distributions.

All translations, derivative works, or aggregate works incorporating any HOWTO documents must be covered under this copyright notice. That is, you may not produce a derivative work from a HOWTO and

impose additional restrictions on its distribution. Exceptions to these rules may be granted under certain conditions; please contact the HOWTO coordinator.

In short, we wish to promote dissemination of this information through as many channels as possible. However, we do wish to retain copyright on the HOWTO documents, and would like to be notified of any plans to redistribute the HOWTOs.

1.3. Where to obtain the important pieces

The official IPF homepage is at: [<http://coombs.anu.edu.au/~avalon/ip-filter.html>](http://coombs.anu.edu.au/~avalon/ip-filter.html)

The most up-to-date version of this document can be found at: [<http://www.obfuscation.org/ipf/>](http://www.obfuscation.org/ipf/)

2. Basic Firewalling

This section is designed to familiarize you with ipfilter's syntax, and firewall theory in general. The features discussed here are features you'll find in any good firewall package. This section will give you a good foundation to make reading and understanding the advanced section very easy. It must be emphasized that this section alone is not enough to build a good firewall, and that the advanced section really is required reading for anybody who wants to build an effective security system.

2.1. Config File Dynamics, Order and Precedence

IPF (IP Filter) has a config file (as opposed to say, running some command again and again for each new rule). The config file drips with Unix: There's one rule per line, the "#" mark denotes a comment, and you can have a rule and a comment on the same line. Extraneous whitespace is allowed, and is encouraged to keep the rules readable.

2.2. Basic Rule Processing

The rules are processed from top to bottom, each one appended after another. This quite simply means that if the entirety of your config file is:

```
block in all
pass in all
```

The computer sees it as:

```
block in all
pass in all
```

Which is to say that when a packet comes in, the first thing IPF applies is:

```
block in all
```

Should IPF deem it necessary to move on to the next rule, it would then apply the second rule:

```
pass in all
```

At this point, you might want to ask yourself "would IPF move on to the second rule?" If you're familiar with ipfwadm or ipfw, you probably won't ask yourself this. Shortly after, you will become bewildered at the weird way packets are always getting denied or passed when they shouldn't. Many packet filters stop comparing packets to rulesets the moment the first match is made; IPF is not one of them.

Unlike the other packet filters, IPF keeps a flag on whether or not it's going to pass the packet. Unless you interrupt the flow, IPF will go through the entire ruleset, making its decision on whether or not to pass or drop the packet based on the last matching rule. The scene: IP Filter's on duty. It's been been scheduled a slice of CPU time. It has a checkpoint clipboard that reads:

```
block in all
pass  in all
```

A packet comes in the interface and it's time to go to work. It takes a look at the packet, it takes a look at the first rule:

```
block in all
```

"So far I think I will block this packet" says IPF. It takes a look at the second rule:

```
pass  in all
```

"So far I think I will pass this packet" says IPF. It takes a look at a third rule. There is no third rule, so it goes with what its last motivation was, to pass the packet onward.

It's a good time to point out that even if the ruleset had been

```
block in all
block in all
block in all
block in all
pass  in all
```

that the packet would still have gone through. There is no cumulative effect. The last matching rule always takes precedence.

2.3. Controlling Rule Processing

If you have experience with other packet filters, you may find this layout to be confusing, and you may be speculating that there are problems with portability with other filters and speed of rule matching. Imagine if you had 100 rules and most of the applicable ones were the first 10. There would be a terrible overhead for every packet coming in to go through 100 rules every time. Fortunately, there is a simple keyword you can add to any rule that makes it take action at that match. That keyword is **quick**.

Here's a modified copy of the original ruleset using the **quick** keyword:

```
block in quick all
pass  in      all
```

In this case, IPF looks at the first rule:

```
block in quick all
```

The packet matches and the search is over. The packet is expunged without a peep. There are no notices, no logs, no memorial service. Cake will not be served. So what about the next rule?

```
pass  in      all
```

This rule is never encountered. It could just as easily not be in the config file at all. The sweeping match of **all** and the terminal keyword **quick** from the previous rule make certain that no rules are followed afterward.

Having half a config file laid to waste is rarely a desirable state. On the other hand, IPF is here to block packets and as configured, it's doing a very good job. Nonetheless, IPF is also here to let some packets through, so a change to the ruleset to make this possible is called for.

2.4. Basic filtering by IP address

IPF will match packets on many criteria. The one that we most commonly think of is the IP address. There are some blocks of address space from which we should never get traffic. One such block is from the unroutable networks, 192.168.0.0/16 (/16 is the CIDR notation for a netmask. You may be more familiar with the dotted decimal format, 255.255.0.0. IPF accepts both). If you wanted to block 192.168.0.0/16, this is one way to do it:

```
block in quick from 192.168.0.0/16 to any
pass  in      all
```

Now we have a less stringent ruleset that actually does something for us. Lets imagine a packet comes in from 1.2.3.4. The first rule is applied:

```
block in quick from 192.168.0.0/16 to any
```

The packet is from 1.2.3.4, not 192.168.*.*, so there is no match. The second rule is applied:

```
pass in      all
```

The packet from 1.2.3.4 is definitely a part of `all`, so the packet is sent to whatever it's destination happened to be.

On the other hand, suppose we have a packet that comes in from 192.168.1.2. The first rule is applied:

```
block in quick from 192.168.0.0/16 to any
```

There's a match, the packet is dropped, and that's the end. Again, it doesn't move to the second rule because the first rule matches and contains the `quick` keyword.

At this point you can build a fairly extensive set of definitive addresses which are passed or blocked. Since we've already started blocking private address space from entering our firewall, let's take care of the rest of it:

```
block in quick from 192.168.0.0/16 to any
block in quick from 172.16.0.0/12 to any
block in quick from 10.0.0.0/8 to any
pass in      all
```

The first three address blocks are the unroutable IP space.[†]

2.5. Controlling Your Interfaces

It seems very frequent that companies have internal networks before they want a link to the outside world. In fact, it's probably reasonable to say that's the main reason people consider firewalls in the first place. The machine that bridges the outside world to the inside world and vice versa is the router. What separates the router from any other machine is simple: It has more than one interface.

Every packet you receive comes from a network interface; every packet you transmit goes out a network interface. Say your machine has 3 interfaces, `lo0` (loopback), `xl0` (3com ethernet), and `tun0` (FreeBSD's generic tunnel interface that PPP uses), but you don't want packets coming in on the `tun0` interface?

```
block in quick on tun0 all
pass in      all
```

In this case, the `on` keyword means that that data is coming in on the named interface. If a packet comes `in on tun0`, the first rule will block it. If a packet comes `in on lo0` or `in on xl0`, the first rule will not match, the second rule will, the packet will be passed.

2.6. Using IP Address and Interface Together

It's an odd state of affairs when one decides it best to have the `tun0` interface up, but not allow any data to be received from it. The more criteria the firewall matches against, the tighter (or looser) the firewall can become. Maybe you want data from `tun0`, but not from 192.168.0.0/16? This is the start of a powerful firewall.

```
block in quick on tun0 from 192.168.0.0/16 to any
pass in      all
```

Compare this to our previous rule:

```
block in quick from 192.168.0.0/16 to any
pass in      all
```

The old way, all traffic from 192.168.0.0/16, regardless of interface, was completely blocked. The new way, using `on tun0` means that it's only blocked if it comes in on the `tun0` interface. If a packet arrived on the `xl0` interface from 192.168.0.0/16, it would be passed.

[†] See rfc1918 at <<http://www.faqs.org/rfcs/rfc1918.html>>

At this point you can build a fairly extensive set of definitive addresses which are passed or blocked. Since we've already started blocking private address space from entering `tun0`, let's take care of the rest of it:

```
block in quick on tun0 from 192.168.0.0/16 to any
block in quick on tun0 from 172.16.0.0/12 to any
block in quick on tun0 from 10.0.0.0/8 to any
block in quick on tun0 from 127.0.0.0/8 to any
pass in all
```

You've already seen the first 3 `blocks`, but not the fourth. The fourth is a largely wasted class-A network used for loopback. Much software communicates with itself on 127.0.0.1 so blocking it from an external source is a good idea.

There's a very important principle in packet filtering which has only been alluded to with the private network blocking and that is this: When you know there's certain types of data that only comes from certain places, you setup the system to only allow that kind of data from those places. In the case of the unroutable addresses, you know that nothing from 10.0.0.0/8 should be arriving on `tun0` because you have no way to reply to it. It's an illegitimate packet. The same goes for the other unroutables as well as 127.0.0.0/8.

Many pieces of software do all their authentication based upon the packet's originating IP address. When you have an internal network, say 20.20.20.0/24, you know that the only traffic for that internal network is going to come off the local ethernet. Should a packet from 20.20.20.0/24 arrive over a PPP dialup, it's perfectly reasonable to drop it on the floor, or put it in a dark room for interrogation. It should by no means be allowed to get to its final destination. You can accomplish this particularly easily with what you already know of IPF. The new ruleset would be:

```
block in quick on tun0 from 192.168.0.0/16 to any
block in quick on tun0 from 172.16.0.0/12 to any
block in quick on tun0 from 10.0.0.0/8 to any
block in quick on tun0 from 127.0.0.0/8 to any
block in quick on tun0 from 20.20.20.0/24 to any
pass in all
```

2.7. Bi-Directional Filtering; The "out" Keyword

Up until now, we've been passing or blocking inbound traffic. To clarify, inbound traffic is all traffic that enters the firewall on any interface. Conversely, outbound traffic is all traffic that leaves on any interface (whether locally generated or simply passing through). This means that all packets coming in are not only filtered as they enter the firewall, they're also filtered as they exit. Thus far there's been an implied `pass out all` that may or may not be desirable.[†] Just as you may pass and block incoming traffic, you may do the same with outgoing traffic.

Now that we know there's a way to filter outbound packets just like inbound, it's up to us to find a conceivable use for such a thing. One possible use of this idea is to keep spoofed packets from exiting your own network. Instead of passing any traffic out the router, you could instead limit permitted traffic to packets originating at 20.20.20.0/24. You might do it like this:

```
pass out quick on tun0 from 20.20.20.0/24 to any
block out quick on tun0 from any to any
```

If a packet comes from 20.20.20.1/32, it gets sent out by the first rule. If a packet comes from 1.2.3.4/32 it gets blocked by the second.

You can also make similar rules for the unroutable addresses. If some machine tries to route a packet through IPF with a destination in 192.168.0.0/16, why not drop it? The worst that can happen is that you'll spare yourself some bandwidth:

```
block out quick on tun0 from any to 192.168.0.0/16
block out quick on tun0 from any to 172.16.0.0/12
block out quick on tun0 from any to 10.0.0.0/8
```

In the narrowest viewpoint, this doesn't enhance your security. It enhances everybody else's security, and that's a nice thing to do. As another viewpoint, one might suppose that because nobody can send spoofed

[†] This can, of course, be changed by using `-DIPFILTER_DEFAULT_BLOCK` when compiling `ipfilter` on your system.

packets from your site, that your site has less value as a relay for crackers, and as such is less of a target.

You'll likely find a number of uses for blocking outbound packets. One thing to always keep in mind is that in and out directions are in reference to your firewall, never any other machine.

2.8. Logging What Happens; The "log" Keyword

Up to this point, all blocked and passed packets have been silently blocked and silently passed. Usually you want to know if you're being attacked rather than wonder if that firewall is really buying you any added benefits. While I wouldn't want to log every passed packet, and in some cases every blocked packet, I would want to know about the blocked packets from 20.20.20.0/24. To do this, we add the `log` keyword:

```
block in    quick on tun0 from 192.168.0.0/16 to any
block in    quick on tun0 from 172.16.0.0/12 to any
block in    quick on tun0 from 10.0.0.0/8 to any
block in    quick on tun0 from 127.0.0.0/8 to any
block in log quick on tun0 from 20.20.20.0/24 to any
pass in     all
```

So far, our firewall is pretty good at blocking packets coming to it from suspect places, but there's still more to be done. For one thing, we're accepting packets destined anywhere. One thing we ought to do is make sure packets to 20.20.20.0/32 and 20.20.20.255/32 get dropped on the floor. To do otherwise opens the internal network for a smurf attack. These two lines would prevent our hypothetical network from being used as a smurf relay:

```
block in log quick on tun0 from any to 20.20.20.0/32
block in log quick on tun0 from any to 20.20.20.255/32
```

This brings our total ruleset to look something like this:

```
block in    quick on tun0 from 192.168.0.0/16 to any
block in    quick on tun0 from 172.16.0.0/12 to any
block in    quick on tun0 from 10.0.0.0/8 to any
block in    quick on tun0 from 127.0.0.0/8 to any
block in log quick on tun0 from 20.20.20.0/24 to any
block in log quick on tun0 from any to 20.20.20.0/32
block in log quick on tun0 from any to 20.20.20.255/32
pass in     all
```

2.9. Complete Bi-Directional Filtering By Interface

So far we have only presented fragments of a complete ruleset. When you're actually creating your ruleset, you should setup rules for every direction and every interface. The default state of `ipfilter` is to pass packets. It is as though there were an invisible rule at the beginning which states `pass in all` and `pass out all`. Rather than rely on some default behaviour, make everything as specific as possible, interface by interface, until every base is covered.

First we'll start with the `lo0` interface, which wants to run wild and free. Since these are programs talking to other on the local system, go ahead and keep it unrestricted:

```
pass out quick on lo0
pass in  quick on lo0
```

Next, there's the `xl0` interface. Later on we'll begin placing restrictions on the `xl0` interface, but to start with, we'll act as though everything on our local network is trustworthy and give it much the same treatment as `lo0`:

```
pass out quick on xl0
pass in  quick on xl0
```

Finally, there's the `tun0` interface, which we've been half-filtering with up until now:

```
block out quick on tun0 from any to 192.168.0.0/16
block out quick on tun0 from any to 172.16.0.0/12
block out quick on tun0 from any to 10.0.0.0/8
pass  out quick on tun0 from 20.20.20.0/24 to any
block out quick on tun0 from any to any

block in    quick on tun0 from 192.168.0.0/16 to any
```

```
block in    quick on tun0 from 172.16.0.0/12 to any
block in    quick on tun0 from 10.0.0.0/8 to any
block in    quick on tun0 from 127.0.0.0/8 to any
block in log quick on tun0 from 20.20.20.0/24 to any
block in log quick on tun0 from any to 20.20.20.0/32
block in log quick on tun0 from any to 20.20.20.255/32
pass in     all
```

This is a pretty significant amount of filtering already, protecting 20.20.20.0/24 from being spoofed or being used for spoofing. Future examples will continue to show one-sidedness, but keep in mind that it's for brevity's sake, and when setting up your own ruleset, adding rules for every direction and every interface is necessary.

2.10. Controlling Specific Protocols; The "proto" Keyword

Denial of Service attacks are as rampant as buffer overflow exploits. Many denial of service attacks rely on glitches in the OS's TCP/IP stack. Frequently, this has come in the form of ICMP packets. Why not block them entirely?

```
block in log quick on tun0 proto icmp from any to any
```

Now any ICMP traffic coming in from `tun0` will be logged and discarded.

2.11. Filtering ICMP with the "icmp-type" Keyword; Merging Rulesets

Of course, dropping all ICMP isn't really an ideal situation. Why not drop all ICMP? Well, because it's useful to have partially enabled. So maybe you want to keep some types of ICMP traffic and drop other kinds. If you want ping and traceroute to work, you need to let in ICMP types 0 and 11. Strictly speaking, this might not be a good idea, but if you need to weigh security against convenience, IPF lets you do it.

```
pass in quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 0
pass in quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 11
```

Remember that ruleset order is important. Since we're doing everything `quick` we must have our `passes` before our `blocks`, so we really want the last three rules in this order:

```
pass in    quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 0
pass in    quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 11
block in log quick on tun0 proto icmp from any to any
```

Adding these 3 rules to the anti-spoofing rules is a bit tricky. One error might be to put the new ICMP rules at the beginning:

```
pass in    quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 0
pass in    quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 11
block in log quick on tun0 proto icmp from any to any
block in    quick on tun0 from 192.168.0.0/16 to any
block in    quick on tun0 from 172.16.0.0/12 to any
block in    quick on tun0 from 10.0.0.0/8 to any
block in    quick on tun0 from 127.0.0.0/8 to any
block in log quick on tun0 from 20.20.20.0/24 to any
block in log quick on tun0 from any to 20.20.20.0/32
block in log quick on tun0 from any to 20.20.20.255/32
pass in     all
```

The problem with this is that an ICMP type 0 packet from 192.168.0.0/16 will get passed by the first rule, and never blocked by the fourth rule. Also, since we quickly pass an ICMP ECHO_REPLY (type 0) to 20.20.20.0/24, we've just opened ourselves back up to a nasty smurf attack and nullified those last two block rules. Oops. To avoid this, we place the ICMP rules after the anti-spoofing rules:

```
block in    quick on tun0 from 192.168.0.0/16 to any
block in    quick on tun0 from 172.16.0.0/12 to any
block in    quick on tun0 from 10.0.0.0/8 to any
block in    quick on tun0 from 127.0.0.0/8 to any
block in log quick on tun0 from 20.20.20.0/24 to any
block in log quick on tun0 from any to 20.20.20.0/32
block in log quick on tun0 from any to 20.20.20.255/32
pass in    quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 0
pass in    quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 11
```

```
block in log quick on tun0 proto icmp from any to any
pass in      all
```

Because we block spoofed traffic before the ICMP rules are processed, a spoofed packet never makes it to the ICMP ruleset. It's very important to keep such situations in mind when merging rules.

2.12. TCP and UDP Ports; The "port" Keyword

Now that we've started blocking packets based on protocol, we can start blocking packets based on specific aspects of each protocol. The most frequently used of these aspects is the port number. Services such as rsh, rlogin, and telnet are all very convenient to have, but also hideously insecure against network sniffing and spoofing. One great compromise is to only allow the services to run internally, then block them externally. This is easy to do because rlogin, rsh, and telnet use specific TCP ports (513, 514, and 23 respectively). As such, creating rules to block them is easy:

```
block in log quick on tun0 proto tcp from any to 20.20.20.0/24 port = 513
block in log quick on tun0 proto tcp from any to 20.20.20.0/24 port = 514
block in log quick on tun0 proto tcp from any to 20.20.20.0/24 port = 23
```

Make sure all 3 are before the `pass in all` and they'll be closed off from the outside (leaving out spoofing for brevity's sake):

```
pass in      quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 0
pass in      quick on tun0 proto icmp from any to 20.20.20.0/24 icmp-type 11
block in log quick on tun0 proto icmp from any to any
block in log quick on tun0 proto tcp from any to 20.20.20.0/24 port = 513
block in log quick on tun0 proto tcp from any to 20.20.20.0/24 port = 514
block in log quick on tun0 proto tcp from any to 20.20.20.0/24 port = 23
pass in      all
```

You might also want to block 514/udp (syslog), 111/tcp & 111/udp (portmap), 515/tcp (lpd), 2049/tcp and 2049/udp (NFS), 6000/tcp (X11) and so on and so forth. You can get a complete listing of the ports being listened to by using `netstat -a` (or `lsof -i`, if you have it installed).

Blocking UDP instead of TCP only requires replacing `proto tcp` with `proto udp`. The rule for syslog would be:

```
block in log quick on tun0 proto udp from any to 20.20.20.0/24 port = 514
```

IPF also has a shorthand way to write rules that apply to both `proto tcp` and `proto udp` at the same time, such as portmap or NFS. The rule for portmap would be:

```
block in log quick on tun0 proto tcp/udp from any to 20.20.20.0/24 port = 111
```

3. Advanced Firewalling Introduction

This section is designed as an immediate followup to the basic section. Contained below are both concepts for advanced firewall design, and advanced features contained only within ipfilter. Once you are comfortable with this section, you should be able to build a very strong firewall.

3.1. Rampant Paranoia; or The Default-Deny Stance

There's a big problem with blocking services by the port: sometimes they move. RPC based programs are terrible about this, lockd, statd, even nfsd listens places other than 2049. It's awfully hard to predict, and even worse to automate adjusting all the time. What if you miss a service? Instead of dealing with all that hassle, lets start over with a clean slate. The current ruleset looks like this:

Yes, we really are starting over. The first rule we're going to use is this:

```
block in all
```

No network traffic gets through. None. Not a peep. You're rather secure with this setup. Not terribly useful, but quite secure. The great thing is that it doesn't take much more to make your box rather secure, yet useful too. Lets say the machine this is running on is a web server, nothing more, nothing less. It doesn't even do DNS lookups. It just wants to take connections on 80/tcp and that's it. We can do that. We can do that with a second rule, and you already know how:

```
block in      on tun0 all
pass  in quick on tun0 proto tcp from any to 20.20.20.1/32 port = 80
```

This machine will pass in port 80 traffic for 20.20.20.1, and deny everything else. For basic firewalling, this is all one needs.

3.2. Implicit Allow; The "keep state" Rule

The job of your firewall is to prevent unwanted traffic getting to point B from point A. We have general rules which say "as long as this packet is to port 23, it's okay." We have general rules which say "as long as this packet has its FIN flag set, it's okay." Our firewalls don't know the beginning, middle, or end of any TCP/UDP/ICMP session. They merely have vague rules that are applied to all packets. We're left to hope that the packet with its FIN flag set isn't really a FIN scan, mapping our services. We hope that the packet to port 23 isn't an attempted hijack of our telnet session. What if there was a way to identify and authorize individual TCP/UDP/ICMP sessions and distinguish them from port scanners and DoS attacks? There is a way, it's called keeping state.

We want convenience and security in one. Lots of people do, that's why Ciscos have an "established" clause that lets established tcp sessions go through. Ipfw has established. Ipfwadm has setup/established. They all have this feature, but the name is very misleading. When we first saw it, we thought it meant our packet filter was keeping track of what was going on, that it knew if a connection was really established or not. The fact is, they're all taking the packet's word for it from a part of the packet anybody can lie about. They read the TCP packet's flags section and there's the reason UDP/ICMP don't work with it, they have no such thing. Anybody who can create a packet with bogus flags can get by a firewall with this setup.

Where does IPF come in to play here, you ask? Well, unlike the other firewalls, IPF really can keep track of whether or not a connection is established. And it'll do it with TCP, UDP and ICMP, not just TCP. Ipfcalls it keeping state. The keyword for the ruleset is `keep state`.

Up until now, we've told you that packets come in, then the ruleset gets checked; packets go out, then the ruleset gets checked. Actually, what happens is packets come in, the state table gets checked, then **maybe** the inbound ruleset gets checked; packets go out, the state table gets checked, then **maybe** the outbound ruleset gets checked. The state table is a list of TCP/UDP/ICMP sessions that are unquestionably passed through the firewall, circumventing the entire ruleset. Sound like a serious security hole? Hang on, it's the best thing that ever happened to your firewall.

All TCP/IP sessions have a start, a middle, and an end (even though they're sometimes all in the same packet). You can't have an end without a middle and you can't have a middle without a start. This means that all you really need to filter on is the beginning of a TCP/UDP/ICMP session. If the beginning of the session is allowed by your firewall rules, you really want the middle and end to be allowed too (lest your IP stack should overflow and your machines become useless). Keeping state allows you to ignore the middle and end and simply focus on blocking/passing new sessions. If the new session is passed, all its subsequent packets will be allowed through. If it's blocked, none of its subsequent packets will be allowed through. Here's an example for running an ssh server (and nothing but an ssh server):

```
block out quick on tun0 all
pass  in  quick on tun0 proto tcp from any to 20.20.20.1/32 port = 22 keep state
```

The first thing you might notice is that there's no "pass out" provision. In fact, there's only an all-inclusive "block out" rule. Despite this, the ruleset is complete. This is because by keeping state, the entire ruleset is circumvented. Once the first SYN packet hits the ssh server, state is created and the remainder of the ssh session is allowed to take place without interference from the firewall. Here's another example:

```
block in  quick on tun0 all
pass  out quick on tun0 proto tcp from 20.20.20.1/42 to any keep state
```

In this case, the server is running no services. Infact, it's not a server, it's a client. And this client doesn't want unauthorized packets entering its IP stack at all. However, the client wants full access to the internet and the reply packets that such privilege entails. This simple ruleset creates state entries for every new outgoing TCP session. Again, since a state entry is created, these new TCP sessions are free to talk back and forth as they please without the hinderance or inspection of the firewall ruleset. We mentioned that this also works for UDP and ICMP:

```
block in quick on tun0 all
pass out quick on tun0 proto tcp from 20.20.20.1/42 to any keep state
pass out quick on tun0 proto udp from 20.20.20.1/42 to any keep state
pass out quick on tun0 proto icmp from 20.20.20.1/42 to any keep state
```

Yes Virginia, we can ping. Now we're keeping state on TCP, UDP, ICMP. Now we can make outgoing connections as though there's no firewall at all, yet would-be attackers can't get back in. This is very handy because there's no need to track down what ports we're listening to, only the ports we want people to be able to get to.

State is pretty handy, but it's also a bit tricky. You can shoot yourself in the foot in strange and mysterious ways. Consider the following ruleset:

```
pass in quick on tun0 proto tcp from any to 20.20.20.1/32 port = 23
pass out quick on tun0 proto tcp from any to any keep state
block in quick all
block out quick all
```

At first glance, this seems to be a good setup. We allow incoming sessions to port 23, and outgoing sessions anywhere. Naturally packets going to port 23 will have reply packets, but the ruleset is setup in such a way that the pass out rule will generate a state entry and everything will work perfectly. At least, you'd think so.

The unfortunate truth is that after 60 seconds of idle time the state entry will be closed (as opposed to the normal 5 days). This is because the state tracker never saw the original SYN packet destined to port 23, it only saw the SYN ACK. IPF is very good about following TCP sessions from start to finish, but it's not very good about coming into the middle of a connection, so rewrite the rule to look like this:

```
pass in quick on tun0 proto tcp from any to 20.20.20.1/32 port = 23 keep state
pass out quick on tun0 proto tcp from any to any keep state
block in quick all
block out quick all
```

The additional of this rule will enter the very first packet into the state table and everything will work as expected. Once the 3-way handshake has been witness by the state engine, it is marked in 4/4 mode, which means it's setup for long-term data exchange until such time as the connection is torn down (wherein the mode changes again. You can see the current modes of your state table with `ipfstat -s`.

3.3. Stateful UDP

UDP is stateless so naturally it's a bit harder to do a reliable job of keeping state on it. Nonetheless, ipf does a pretty good job. When machine A sends a UDP packet to machine B with source port X and destination port Y, ipf will allow a reply from machine B to machine A with source port Y and destination port X. This is a short term state entry, a mere 60 seconds.

Here's an example of what happens if we use nslookup to get the IP address of www.3com.com:

```
$ nslookup www.3com.com
```

A DNS packet is generated:

```
17:54:25.499852 20.20.20.1.2111 > 198.41.0.5.53: 51979+
```

The packet is from 20.20.20.1, port 2111, destined for 198.41.0.5, port 53. A 60 second state entry is created. If a packet comes back from 198.41.0.5 port 53 destined for 20.20.20.1 port 2111 within that period of time, the reply packet will be let through. As you can see, milliseconds later:

```
17:54:25.501209 198.41.0.5.53 > 20.20.20.1.2111: 51979 q: www.3com.com
```

The reply packet matches the state criteria and is let through. At that same moment that packet is let through, the state gateway is closed and no new incoming packets will be allowed in, even if they claim to be from the same place.

3.4. Stateful ICMP

IPFilter handles ICMP states in the manner that one would expect from understanding how ICMP is used with TCP and UDP, and with your understanding of how `keep state` works. There are two general types of ICMP messages; requests and replies. When you write a rule such as:

```
pass out on tun0 proto icmp from any to any icmp-type 8 keep state
```

to allow outbound echo requests (a typical ping), the resultant icmp-type 0 packet that comes back will be allowed in. This state entry has a default timeout of an incomplete 0/0 state of 60 seconds. Thus, if you are keeping state on any outbound icmp message that will elicit an icmp message in reply, you need a `proto icmp [...] keep state` rule.

However, the majority of ICMP messages are status messages generated by some failure in UDP (and sometimes TCP), and in 3.4.x and greater IPFilters, any ICMP error status message (say `icmp-type 3 code 3` port unreachable, or `icmp-type 11` time exceeded) that matches an active state table entry that could have generated that message, the ICMP packet is let in. For example, in older IPFilters, if you wanted traceroute to work, you needed to use:

```
pass out on tun0 proto udp from any to any port 33434><33690 keep state
pass in on tun0 proto icmp from any to any icmp-type timex
```

whereas now you can do the right thing and just keep state on udp with:

```
pass out on tun0 proto udp from any to any port 33434><33690 keep state
```

To provide some protection against a third-party sneaking ICMP messages through your firewall when an active connection is known to be in your state table, the incoming ICMP packet is checked not only for matching source and destination addresses (and ports, when applicable) but a tiny part of the payload of the packet that the ICMP message is claiming it was generated by.

3.5. FIN Scan Detection; "flags" Keyword, "keep frags" Keyword

Lets go back to the 4 rule set from the previous section:

```
pass in quick on tun0 proto tcp from any to 20.20.20.1/32 port = 23 keep state
pass out quick on tun0 proto tcp from any to any keep state
block in quick all
block out quick all
```

This is almost, but not quite, satisfactory. The problem is that it's not just SYN packets that're allowed to go to port 23, any old packet can get through. We can change this by using the `flags` option:

```
pass in quick on tun0 proto tcp from any to 20.20.20.1/32 port = 23 flags S keep state
pass out quick on tun0 proto tcp from any to any flags S keep state
block in quick all
block out quick all
```

Now only TCP packets, destined for 20.20.20.1, at port 23, with a lone SYN flag will be allowed in and entered into the state table. A lone SYN flag is only present as the very first packet in a TCP session (called the TCP handshake) and that's really what we wanted all along. There's at least two advantages to this: No arbitrary packets can come in and make a mess of your state table. Also, FIN and XMAS scans will fail since they set flags other than the SYN flag.[†] Now all incoming packets must either be handshakes or have state already. If anything else comes in, it's probably a port scan or a forged packet. There's one exception to that, which is when a packet comes in that's fragmented from its journey. IPF has provisions for this as well, the `keep frags` keyword. With it, IPF will notice and keep track of packets that are fragmented,

[†] Some examples use `flags S/SA` instead of `flags S`. `flags S` actually equates to `flags S/AUPRFS` and matches against only the SYN packet out of all six possible flags, while `flags S/SA` will allow packets that may or may not have the URG, PSH, FIN, or RST flags set. Some protocols demand the URG or PSH flags, and `S/SAFR` would be a better choice for these, however we feel that it is less secure to blindly use `S/SA` when it isn't required. But it's your firewall.

allowing the expected fragments to go through. Lets rewrite the 3 rules to log forgeries and allow fragments:

```
pass in quick on tun0 proto tcp from any to 20.20.20.1/32 port = 23 flags S keep state keep frags
pass out quick on tun0 proto tcp from any to any keep state flags S keep frags
block in log quick all
block out log quick all
```

This works because every packet that should be allowed through makes it into the state table before the blocking rules are reached. The only scan this won't detect is a SYN scan itself. If you're truly worried about that, you might even want to log all initial SYN packets.

3.6. Responding To a Blocked Packet

So far, all of our blocked packets have been dumped on the floor, logged or not, we've never sent anything back to the originating host. Sometimes this isn't the most desirable of responses because in doing so, we actually tell the attacker that a packet filter is present. It seems a far better thing to misguide the attacker into believing that, while there's no packet filter running, there's likewise no services to break into. This is where fancier blocking comes into play.

When a service isn't running on a Unix system, it normally lets the remote host know with some sort of return packet. In TCP, this is done with an RST (Reset) packet. When blocking a TCP packet, IPF can actually return an RST to the origin by using the `return-rst` keyword.

Where once we did:

```
block in log on tun0 proto tcp from any to 20.20.20.0/24 port = 23
pass in all
```

We might now do:

```
block return-rst in log from any to 20.20.20.0/24 proto tcp port = 23
block in log quick on tun0
pass in all
```

We need two `block` statements since `return-rst` only works with TCP, and we still want to block protocols such as UDP, ICMP, and others. Now that this is done, the remote side will get "connection refused" instead of "connection timed out".

It's also possible to send an error message when somebody sends a packet to a UDP port on your system. Whereas once you might have used:

```
block in log quick on tun0 proto udp from any to 20.20.20.0/24 port = 111
```

You could instead use the `return-icmp` keyword to send a reply:

```
block return-icmp(port-unr) in log quick on tun0 proto udp from any to 20.20.20.0/24 port = 111
```

According to *TCP/IP Illustrated*, port-unreachable is the correct ICMP type to return when no service is listening on the port in question. You can use any ICMP type you like, but port-unreachable is probably your best bet. It's also the default ICMP type for `return-icmp`.

However, when using `return-icmp`, you'll notice that it's not very stealthy, and it returns the ICMP packet with the IP address of the firewall, not the original destination of the packet. This was fixed in ipfilter 3.3, and a new keyword; `return-icmp-as-dest`, has been added. The new format is:

```
block return-icmp-as-dest(port-unr) in log on tun0 proto udp from any to 20.20.20.0/24 port = 111
```

3.7. Fancy Logging Techniques

It is important to note that the presence of the `log` keyword only ensures that the packet will be available to the ipfilter logging device; `/dev/ipl`. In order to actually see this log information, one must be running the `ipmon` utility (or some other utility that reads from `/dev/ipl`). The typical usage of `log` is coupled with `ipmon -s` to log the information to syslog. As of ipfilter 3.3, one can now even control the logging behavior of syslog by using `log level` keywords, as in rules such as this:

```
block in log level auth.info quick on tun0 from 20.20.20.0/24 to any
block in log level auth.alert quick on tun0 proto tcp from any to 20.20.20.0/24 port = 21
```

In addition to this, you can tailor what information is being logged. For example, you may not be interested that someone attempted to probe your telnet port 500 times, but you are interested that they probed you once. You can use the `log first` keyword to only log the first example of a packet. Of course, the notion of "first-ness" only applies to packets in a specific session, and for the typical blocked packet, you will be hard pressed to encounter situations where this does what you expect. However, if used in conjunction with `pass` and `keep state`, this can be a valuable keyword for keeping tabs on traffic.

Another useful thing you can do with the logs is to keep track of interesting pieces of the packet in addition to the header information normally being logged. `Ipfilter` will give you the first 128 bytes of the packet if you use the `log body` keyword. You should limit the use of body logging, as it makes your logs very verbose, but for certain applications, it is often handy to be able to go back and take a look at the packet, or to send this data to another application that can examine it further.

3.8. Putting It All Together

So now we have a pretty tight firewall, but it can still be tighter. Some of the original ruleset we wiped clean is actually very useful. I'd suggest bringing back all the anti-spoofing stuff. This leaves us with:

```
block in      on tun0
block in      quick on tun0 from 192.168.0.0/16 to any
block in      quick on tun0 from 172.16.0.0/12 to any
block in      quick on tun0 from 10.0.0.0/8 to any
block in      quick on tun0 from 127.0.0.0/8 to any
block in log   quick on tun0 from 20.20.20.0/24 to any
block in log   quick on tun0 from any to 20.20.20.0/32
block in log   quick on tun0 from any to 20.20.20.255/32
pass out      quick on tun0 proto tcp/udp from 20.20.20.1/32 to any keep state
pass out      quick on tun0 proto icmp   from 20.20.20.1/32 to any keep state
pass in       quick on tun0 proto tcp from any to 20.20.20.1/32 port = 80 flags S keep state
```

3.9. Improving Performance With Rule Groups

Let's extend our use of our firewall by creating a much more complicated, and we hope more applicable to the real world, example configuration. For this example, we're going to change the interface names, and network numbers. Let's assume that we have three interfaces in our firewall with interfaces `xl0`, `xl1`, and `xl2`.

```
xl0 is connected to our external network 20.20.20.0/26
xl1 is connected to our "DMZ" network 20.20.20.64/26
xl2 is connected to our protected network 20.20.20.128/25
```

We'll define the entire ruleset in one swoop, since we figure that you can read these rules by now:

```
block in      quick on xl0 from 192.168.0.0/16 to any
block in      quick on xl0 from 172.16.0.0/12 to any
block in      quick on xl0 from 10.0.0.0/8 to any
block in      quick on xl0 from 127.0.0.0/8 to any
block in log   quick on xl0 from 20.20.20.0/24 to any
block in log   quick on xl0 from any to 20.20.20.0/32
block in log   quick on xl0 from any to 20.20.20.63/32
block in log   quick on xl0 from any to 20.20.20.64/32
block in log   quick on xl0 from any to 20.20.20.127/32
block in log   quick on xl0 from any to 20.20.20.128/32
block in log   quick on xl0 from any to 20.20.20.255/32
pass out      on xl0 all
pass out      quick on xl1 proto tcp from any to 20.20.20.64/26 port = 80 flags S keep state
pass out      quick on xl1 proto tcp from any to 20.20.20.64/26 port = 21 flags S keep state
pass out      quick on xl1 proto tcp from any to 20.20.20.64/26 port = 20 flags S keep state
pass out      quick on xl1 proto tcp from any to 20.20.20.65/32 port = 53 flags S keep state
pass out      quick on xl1 proto udp from any to 20.20.20.65/32 port = 53 keep state
pass out      quick on xl1 proto tcp from any to 20.20.20.66/32 port = 53 flags S keep state
pass out      quick on xl1 proto udp from any to 20.20.20.66/32 port = 53 keep state
block out     on xl1 all
pass in       quick on xl1 proto tcp/udp from 20.20.20.64/26 to any keep state
block out     on xl2 all
```

```
pass in quick on xl2 proto tcp/udp from 20.20.20.128/25 to any keep state
```

From this arbitrary example, we can already see that our ruleset is becoming unwieldy. To make matters worse, as we add more specific rules to our DMZ network, we add additional tests that must be parsed for every packet, which affects the performance of the xl0 <-> xl2 connections. If you set up a firewall with a ruleset like this, and you have lots of bandwidth and a moderate amount of cpu, everyone that has a workstation on the xl2 network is going to come looking for your head to place on a platter. So, to keep your head <-> torso network intact, you can speed things along by creating rule groups. Rule groups allow you to write your ruleset in a tree fashion, instead of as a linear list, so that if your packet has nothing to do with the set of tests (say, all those xl1 rules) those rules will never be consulted. It's somewhat like having multiple firewalls all on the same machine.

Here's a simple example to get us started:

```
block out quick on xl1 all head 10
pass out quick proto tcp from any to 20.20.20.64/26 port = 80 flags S keep state group 10
block out on xl2 all
```

In this simplistic example, we can see a small hint of the power of the rule group. If the packet is not destined for xl1, the head of rule group 10 will not match, and we will go on with our tests. If the packet does match for xl1, the quick keyword will short-circuit all further processing at the root level (rule group 0), and focus the testing on rules which belong to group 10; namely, the SYN check for 80/tcp. In this way, we can re-write the above rules so that we can maximize performance of our firewall.

```
block in quick on xl0 all head 1
block in quick on xl0 from 192.168.0.0/16 to any group 1
block in quick on xl0 from 172.16.0.0/12 to any group 1
block in quick on xl0 from 10.0.0.0/8 to any group 1
block in quick on xl0 from 127.0.0.0/8 to any group 1
block in log quick on xl0 from 20.20.20.0/24 to any group 1
block in log quick on xl0 from any to 20.20.20.0/32 group 1
block in log quick on xl0 from any to 20.20.20.63/32 group 1
block in log quick on xl0 from any to 20.20.20.64/32 group 1
block in log quick on xl0 from any to 20.20.20.127/32 group 1
block in log quick on xl0 from any to 20.20.20.128/32 group 1
block in log quick on xl0 from any to 20.20.20.255/32 group 1
pass in on xl0 all group 1

pass out on xl0 all

block out quick on xl1 all head 10
pass out quick on xl1 proto tcp from any to 20.20.20.64/26 port = 80 flags S keep state group 10
pass out quick on xl1 proto tcp from any to 20.20.20.64/26 port = 21 flags S keep state group 10
pass out quick on xl1 proto tcp from any to 20.20.20.64/26 port = 20 flags S keep state group 10
pass out quick on xl1 proto tcp from any to 20.20.20.65/32 port = 53 flags S keep state group 10
pass out quick on xl1 proto udp from any to 20.20.20.65/32 port = 53 keep state group 10
pass out quick on xl1 proto tcp from any to 20.20.20.66/32 port = 53 flags S keep state
pass out quick on xl1 proto udp from any to 20.20.20.66/32 port = 53 keep state group 10

pass in quick on xl1 proto tcp/udp from 20.20.20.64/26 to any keep state

block out on xl2 all

pass in quick on xl2 proto tcp/udp from 20.20.20.128/25 to any keep state
```

Now you can see the rule groups in action. For a host on the xl2 network, we can completely bypass all the checks in group 10 when we're not communicating with hosts on that network.

Depending on your situation, it may be prudent to group your rules by protocol, or various machines, or netblocks, or whatever makes it flow smoothly.

3.10. "Fastroute"; The Keyword of Stealthiness

Even though we're forwarding some packets, and blocking other packets, we're typically behaving like a well behaved router should by decrementing the TTL on the packet and acknowledging to the entire world that yes, there is a hop here. But we can hide our presence from inquisitive applications like unix traceroute which uses UDP packets with various TTL values to map the hops between two sites. If we want incoming traceroutes to work, but we do not want to announce the presence of our firewall as a hop, we can do so with a rule like this:

```
block in quick on xl0 fastroute proto udp from any to any port 33434 >< 33465
```

The presence of the `fastroute` keyword will signal `ipfilter` to not pass the packet into the Unix IP stack for routing which results in a TTL decrement. The packet will be placed gently on the output interface by `ipfilter` itself and no such decrement will happen. `Ipfilter` will of course use the system's routing table to figure out what the appropriate output interface really is, but it will take care of the actual task of routing itself.

There's a reason we used `block quick` in our example, too. If we had used `pass`, and if we had IP Forwarding enabled in our kernel, we would end up having two paths for a packet to come out of, and we would probably panic our kernel.

It should be noted, however, that most Unix kernels (and certainly the ones underlying the systems that `ipfilter` usually runs on) have far more efficient routing code than what exists in `ipfilter`, and this keyword should not be thought of as a way to improve the operating speed of your firewall, and should only be used in places where stealth is an issue.

4. NAT and Proxies

Outside of the corporate environment, one of the biggest enticements of firewall technology to the end user is the ability to connect several computers through a common external interface, often without the approval, knowledge, or even consent of their service provider. To those familiar with Linux, this concept is called *IP Masquerading*, but to the rest of the world it is known by the more obscure name of *Network Address Translation*, or NAT for short.[†]

4.1. Mapping Many Addresses Into One Address

The basic use of NAT accomplishes the same thing that Linux's IP Masquerading function does, and it does it with one simple rule:

```
map tun0 192.168.1.0/24 -> 20.20.20.1/32
```

Very simple. Whenever a packet goes out the `tun0` interface with a source address matching the CIDR network mask of `192.168.1.0/24` ^{††} this packet will be rewritten within the IP stack such that its source address is `20.20.20.1`, and it will be sent on to its original destination. The system also keeps a list of what translated connections are in progress so that it can perform the reverse and remap the response (which will be directed to `20.20.20.1`) to the internal host that really generated the packet.

There is a drawback to the rule we have just written, though. In a large number of cases, we do not happen to know what the IP address of our outside link is (if we're using `tun0` or `ppp0` and a typical ISP) so it makes setting up our NAT tables a chore. Luckily, NAT is smart enough to accept an address of `0/32` as a signal that it needs to go look at what the address of that interface really is and we can rewrite our rule as follows:

```
map tun0 192.168.1.0/24 -> 0/32
```

Now we can load our `ipnat` rules with impunity and connect to the outside world without having to edit anything. You do have to run `ipf -y` to refresh the address if you get disconnected and redial or if your DHCP lease changes, though.

Some of you may be wondering what happens to the source port when the mapping happens. With our current rule, the packet's source port is changed to the first available port on the NAT host. There can be instances where we do not desire this behavior; maybe we have another firewall further upstream we have to pass through, or perhaps we use the NAT host for other things and we wish to limit the amount of

[†] To be pedantic, what `IPFilter` provides is really called NPAT, for Network and Port Address Translation, which means we can change any of the source and destination IP Addresses and their source and destination ports. *True NAT* only allows one to change the addresses.

^{††} This is a typical internal address space, since it's non-routable on the Real Internet it is often used for internal networks. You should still block these packets coming in from the outside world as discussed earlier.

ports that the NAT system can consume. `ipnat` helps us here too with the `portmap` keyword:

```
map tun0 192.168.1.0/24 -> 0/32 portmap tcp/udp 20000:30000
```

Our rule now shoehorns all the translated connections (which can be `tcp`, `udp`, or `tcp/udp`) into the port range of 20000 to 30000.

4.2. Mapping Many Addresses Into a Pool of Addresses

Another common use of NAT is to take a small statically allocated block of addresses and map many computers into this smaller address space. This is easy to accomplish using what you already know about the `map` and `portmap` keywords by writing a rule like so:

```
map tun0 192.168.0.0/16 -> 20.20.20.0/24 portmap tcp/udp 20000:60000
```

Also, there may be instances where a remote application requires that multiple connections all come from the same IP address. We can help with these situations by telling NAT to statically map sessions from a host into the pool of addresses and work some magic to choose a port. This uses the keyword `map-block` as follows:

```
map-block tun0 192.168.1.0/24 -> 20.20.20.0/24
```

4.3. One to One Mappings

Occasionally it is desirable to have a system with one IP address behind the firewall to appear to have a completely different IP address. One example of how this would work would be a lab of computers which are then attached to various networks that are to be put under some kind of test. In this example, you would not want to have to reconfigure the entire lab when you could place a NAT system in front and change the addresses in one simple place. We can do that with the `bimap` keyword, for bidirectional mapping. `Bimap` has some additional protections on it to ensure a known state for the connection, whereas the `map` keyword is designed to allocate an address and a source port and rewrite the packet and go on with life.

```
bimap tun0 192.168.1.1/32 -> 20.20.20.1/32
```

will accomplish the mapping for one host.

4.4. Spoofing Services

Spoofing services? What does that have to do with anything? Plenty. Let's pretend that we have a web server running on 20.20.20.5, and since we've gotten increasingly suspicious of our network security, we desire to not run this server on port 80 since that requires a brief lifespan as the root user. But how do we run it on a less privileged port of 8000 in this world of "anything dot com"? How will anyone find our server? We can use the redirection facilities of NAT to solve this problem by instructing it to remap any connections destined for 20.20.20.5:80 to really point to 20.20.20.5:8000. This uses the `rdr` keyword:

```
rdr tun0 20.20.20.5/32 port 80 -> 192.168.0.5 port 8000
```

We can also specify the protocol here, if we wanted to redirect a UDP service, instead of a TCP service (which is the default). For example, if we had a honeypot on our firewall to impersonate the popular Back Orifice for Windows, we could shovel our entire network into this one place with a simple rule:

```
rdr tun0 20.20.20.0/24 port 31337 -> 127.0.0.1 port 31337 udp
```

Another useful thing that `rdr` provides is the ability to selectively redirect connections from specific hosts. For example, if you had a trusted computer that you wanted to access one version of a service, while you wanted the rest of the world to access a different version, you could accomplish this with the following type of `rdr` ruleset:

```
rdr tun0 from 1.2.3.4/32 20.20.20.5/32 port 80 -> 192.168.0.5 port 8001 tcp  
rdr tun0 20.20.20.5/32 port 80 -> 192.168.0.5 port 8000 tcp
```

An extremely important point must be made about `rdr`: You cannot easily† use this feature as a "reflector". E.g:

```
rdr tun0 20.20.20.5/32 port 80 -> 20.20.20.6 port 80 tcp
```

will not work in the situation where .5 and .6 are on the same LAN segment. The `rdr` function is applied to packets that enter the firewall on the specified interface. When a packet comes in that matches a `rdr` rule, its destination address is then rewritten, it is pushed into `ipf` for filtering, and should it successfully run the gauntlet of filter rules, it is then sent to the unix routing code. Since this packet is still *inbound* on the same interface that it will need to leave the system on to reach a host, the system gets confused. Reflectors don't work. Neither does specifying the address of the interface the packet just came in on. Always remember that `rdr` destinations must exit out of the firewall host on a different interface. ††

4.5. Transparent Proxy Support; Redirection Made Useful

Since you're installing a firewall, you may have decided that it is prudent to use a proxy for many of your outgoing connections so that you can further tighten your filter rules protecting your internal network, or you may have run into a situation that the NAT mapping process does not currently handle properly. This can also be accomplished with a redirection statement:

```
rdr xl0 0.0.0.0/0 port 21 -> 127.0.0.1 port 21
```

This statement says that any packet coming in on the `xl0` interface destined for any address (0.0.0.0/0) on the `ftp` port should be rewritten to connect it with a proxy that is running on the NAT system on port 21.

This specific example of FTP proxying does lead to some complications when used with web browsers or other automatic-login type clients that are unaware of the requirements of communicating with the proxy. There are patches for *TIS Firewall Toolkit's* `ftp-gw` to mate it with the `nat` process so that it can determine where you were trying to go and automatically send you there. Many proxy packages now work in a transparent proxy environment (Squid for example, located at <http://squid.nlanr.net>, works fine.)

This application of the `rdr` keyword is often more useful when you wish to force users to authenticate themselves with the proxy. (For example, you desire your engineers to be able to surf the web, but you would rather not have your call-center staff doing so.)

4.6. Magic Hidden Within NAT; Application Proxies

Since `ipnat` provides a method to rewrite packets as they traverse the firewall, it becomes a convenient place to build in some application level proxies to make up for well known deficiencies of that application and typical firewalls. For example; FTP. We can make our firewall pay attention to the packets going across it and when it notices that it's dealing with an Active FTP session, it can write itself some temporary rules, much like what happens with `keep state`, so that the FTP data connection works. To do this, we use a rule like so:

```
map tun0 192.168.1.0/24 -> 20.20.20.1/32 proxy port ftp ftp/tcp
```

You must always remember to place this `proxy` rule **before** any `portmap` rules, otherwise when `portmap` comes along and matches the packet and rewrites it before the proxy gets a chance to work on it. Remember that `ipnat` rules are first-match.

There also exist proxies for "rcmd" (which we suspect is berkeley `r-` commands which should be forbidden anyway, thus we haven't looked at what this proxy does) and "raudio" for Real Audio PNM streams. Likewise, both of these rules should be put before any `portmap` rules, if you're doing NAT.

† Yes. There is a way to do this. It's so convoluted that I refuse to use it, though. Smart people who require this functionality will transparently redirect into something like TIS plug-gw on 127.0.0.1. Stupid people will set up a dummy loop interface pair and double rewrite.

†† This includes 127.0.0.1, by the way. That's on `lo0`. Neat, huh?

5. Loading and Manipulating Filter Rules; The ipf Utility

IP Filter rules are loaded by using the `ipf` utility. The filter rules can be stored in any file on the system, but typically these rules are stored in `/etc/ipf.rules`, `/usr/local/etc/ipf.rules`, or `/etc/opt/ipf/ipf.rules`.

IP Filter has two sets of rules, the *active set* and the *inactive set*. By default, all operations are performed on the active set. You can manipulate the inactive set by adding `-I` to the `ipf` command line. The two sets can be toggled by using the `-S` command line option. This is very useful for testing new rule sets without wiping out the old rule set.

Rules can also be removed from the list instead of added by using the `-r` command line option, but it is generally a safer idea to flush the rule set that you're working on with `-F` and completely reload it when making changes.

In summary, the easiest way to load a rule set is `ipf -Fa -f /etc/ipf.rules`. For more complicated manipulations of the rule set, please see the `ipf(1)` man page.

6. Loading and Manipulating NAT Rules; The ipnat Utility

NAT rules are loaded by using the `ipnat` utility. The NAT rules can be stored in any file on the system, but typically these rules are stored in `/etc/ipnat.rules`, `/usr/local/etc/ipnat.rules`, or `/etc/opt/ipf/ipnat.rules`.

Rules can also be removed from the list instead of added by using the `-r` command line option, but it is generally a safer idea to flush the rule set that you're working on with `-C` and completely reload it when making changes. Any active mappings are not affected by `-C`, and can be removed with `-F`.

NAT rules and active mappings can be examined with the `-l` command line option.

In summary, the easiest way to load a NAT rule set is `ipnat -CF -f /etc/ipnat.rules`.

7. Monitoring and Debugging

There will come a time when you are interested in what your firewall is actually doing, and `ipfilter` would be incomplete if it didn't have a full suite of status monitoring tools.

7.1. The ipfstat utility

In its simplest form, `ipfstat` displays a table of interesting data about how your firewall is performing, such as how many packets have been passed or blocked, if they were logged or not, how many state entries have been made, and so on. Here's an example of something you might see from running the tool:

```
# ipfstat
input packets:      blocked 99286 passed 1255609 nomatch 14686 counted 0
output packets:     blocked 4200 passed 1284345 nomatch 14687 counted 0
input packets logged: blocked 99286 passed 0
output packets logged: blocked 0 passed 0
packets logged:      input 0 output 0
log failures:         input 3898 output 0
fragment state(in):   kept 0 lost 0
fragment state(out):  kept 0 lost 0
packet state(in):     kept 169364 lost 0
packet state(out):    kept 431395 lost 0
ICMP replies:         0
TCP RSTs sent:        0
Result cache hits(in): 1215208 (out): 1098963
IN Pullups succeeded: 2 failed: 0
OUT Pullups succeeded: 0 failed: 0
Fastroute successes: 0 failures: 0
TCP cksum fails(in): 0 (out): 0
Packet log flags set: (0)
none
```

`ipfstat` is also capable of showing you your current rule list. Using the `-i` or the `-o` flag will show the currently loaded rules for in or out, respectively. Adding a `-h` to this will provide more useful information at the same time by showing you a "hit count" on each rule. For example:

```
# ipfstat -ho
```

```
2451423 pass out on xl0 from any to any
354727 block out on ppp0 from any to any
430918 pass out quick on ppp0 proto tcp/udp from 20.20.20.0/24 to any keep state keep frags
```

From this, we can see that perhaps there's something abnormal going on, since we've got a lot of blocked packets outbound, even with a very permissive `pass out` rule. Something here may warrant further investigation, or it may be functioning perfectly by design. `ipfstat` can't tell you if your rules are right or wrong, it can only tell you what is happening because of your rules.

To further debug your rules, you may want to use the `-n` flag, which will show the rule number next to each rule.

```
# ipfstat -on
@1 pass out on xl0 from any to any
@2 block out on ppp0 from any to any
@3 pass out quick on ppp0 proto tcp/udp from 20.20.20.0/24 to any keep state keep frags
```

The final piece of really interesting information that `ipfstat` can provide us is a dump of the state table. This is done with the `-s` flag:

```
# ipfstat -s
281458 TCP
319349 UDP
0 ICMP
19780145 hits
5723648 misses
0 maximum
0 no memory
1 active
319349 expired
281419 closed
100.100.100.1 -> 20.20.20.1 ttl 864000 pass 20490 pr 6 state 4/4
pkts 196 bytes 17394 987 -> 22 585538471:2213225493 16592:16500
pass in log quick keep state
pkt_flags & b = 2, pkt_options & ffffffff = 0
pkt_security & ffff = 0, pkt_auth & ffff = 0
```

Here we see that we have one state entry for a TCP connection. The output will vary slightly from version to version, but the basic information is the same. We can see in this connection that we have a fully established connection (represented by the 4/4 state. Other states are incomplete and will be documented fully later.) We can see that the state entry has a time to live of 240 hours, which is an absurdly long time, but is the default for an established TCP connection. This TTL counter is decremented every second that the state entry is not used, and will finally result in the connection being purged if it has been left idle. The TTL is also reset to 864000 whenever the state IS used, ensuring that the entry will not time out while it is being actively used. We can also see that we have passed 196 packets consisting of about 17kB worth of data over this connection. We can see the ports for both endpoints, in this case 987 and 22; which means that this state entry represents a connection from 100.100.100.1 port 987 to 20.20.20.1 port 22. The really big numbers in the second line are the TCP sequence numbers for this connection, which helps to ensure that someone isn't easily able to inject a forged packet into your session. The TCP window is also shown. The third line is a synopsis of the implicit rule that was generated by the `keep state` code, showing that this connection is an inbound connection.

7.2. The `ipmon` utility

`ipfstat` is great for collecting snapshots of what's going on on the system, but it's often handy to have some kind of log to look at and watch events as they happen in time. `ipmon` is this tool. `ipmon` is capable of watching the packet log (as created with the `log` keyword in your rules), the state log, or the nat log, or any combination of the three. This tool can either be run in the foreground, or as a daemon which logs to syslog or a file. If we wanted to watch the state table in action, `ipmon -o S` would show this:

```
# ipmon -o S
01/08/1999 15:58:57.836053 STATE:NEW 100.100.100.1,53 -> 20.20.20.15,53 PR udp
01/08/1999 15:58:58.030815 STATE:NEW 20.20.20.15,123 -> 128.167.1.69,123 PR udp
01/08/1999 15:59:18.032174 STATE:NEW 20.20.20.15,123 -> 128.173.14.71,123 PR udp
01/08/1999 15:59:24.570107 STATE:EXPIRE 100.100.100.1,53 -> 20.20.20.15,53 PR udp Pkts 4 Bytes 356
01/08/1999 16:03:51.754867 STATE:NEW 20.20.20.13,1019 -> 100.100.100.10,22 PR tcp
01/08/1999 16:04:03.070127 STATE:EXPIRE 20.20.20.13,1019 -> 100.100.100.10,22 PR tcp Pkts 63 Bytes 4604
```

Here we see a state entry for an external dns request off our nameserver, two xntp pings to well-known time servers, and a very short lived outbound ssh connection.

`ipmon` is also capable of showing us what packets have been logged. For example, when using state, you'll often run into packets like this:

```
# ipmon -o I
15:57:33.803147 ppp0 @0:2 b 100.100.100.103,443 -> 20.20.20.10,4923 PR tcp len 20 1488 -A
```

What does this mean? The first field is obvious, it's a timestamp. The second field is also pretty obvious, it's the interface that this event happened on. The third field `@0:2` is something most people miss. This is the rule that caused the event to happen. Remember `ipfstat -in`? If you wanted to know where this came from, you could look there for rule 2 in rule group 0. The fourth field, the little "b" says that this packet was blocked, and you'll generally ignore this unless you're logging passed packets as well, which would be a little "p" instead. The fifth and sixth fields are pretty self-explanatory, they say where this packet came from and where it was going. The seventh ("PR") and eighth fields tell you the protocol and the ninth field tells you the size of the packet. The last part, the "-A" in this case, tells you the flags that were on the packet; This one was an ACK packet. Why did I mention state earlier? Due to the often laggy nature of the Internet, sometimes packets will be regenerated. Sometimes, you'll get two copies of the same packet, and your state rule which keeps track of sequence numbers will have already seen this packet, so it will assume that the packet is part of a different connection. Eventually this packet will run into a real rule and have to be dealt with. You'll often see the last packet of a session being closed get logged because the `keep state` code has already torn down the connection before the last packet has had a chance to make it to your firewall. This is normal, do not be alarmed. [†] Another example packet that might be logged:

```
12:46:12.470951 xl0 @0:1 S 20.20.20.254 -> 255.255.255.255 PR icmp len 20 9216 icmp 9/0
```

This is a ICMP router discovery broadcast. We can tell by the ICMP type 9/0.

Finally, `ipmon` also lets us look at the NAT table in action.

```
# ipmon -o N
01/08/1999 05:30:02.466114 @2 NAT:RDR 20.20.20.253,113 <- -> 20.20.20.253,113 [100.100.100.13,45816]
01/08/1999 05:30:31.990037 @2 NAT:EXPIRE 20.20.20.253,113 <- -> 20.20.20.253,113 [100.100.100.13,45816] Pkts
```

This would be a redirection to an identd that lies to provide ident service for the hosts behind our NAT, since they are typically unable to provide this service for themselves with ordinary natting.

8. Specific Applications of IP Filter - Things that don't fit, but should be mentioned anyway.

8.1. Keep State With Servers and Flags.

Keeping state is a good thing, but it's quite easy to make a mistake in the direction that you want to `keep state` in. Generally, you want to have a `keep state` keyword on the first rule that interacts with a packet for the connection. One common mistake that is made when mixing state tracking with filtering on flags is this:

```
block in all
pass in quick proto tcp from any to 20.20.20.20/32 port = 23 flags S
pass out all keep state
```

That certainly appears to allow a connection to be created to the telnet server on 20.20.20.20, and the replies to go back. If you try using this rule, you'll see that it does work--Momentarily. Since we're filtering for the SYN flag, the state entry never fully gets completed, and the default time to live for an incomplete state is 60 seconds.

We can solve this by rewriting the rules in one of two ways:

[†] For a technical presentation of the IP Filter stateful inspection engine, please see the white paper *Real Stateful TCP Packet Filtering in IP Filter*, by Guido van Rooij. This paper may be found at <http://www.iae.nl/users/guido/papers/tcp_filtering.ps.gz>

1)

```
block in all
pass in quick proto tcp from any to 20.20.20.20/32 port = 23 keep state
block out all
```

or:

2)

```
block in all
pass in quick proto tcp from any to 20.20.20.20/32 port = 23 flags S keep state
pass out all keep state
```

Either of these sets of rules will result in a fully established state entry for a connection to your server.

8.2. Coping With FTP

FTP is one of those protocols that you just have to sit back and ask "What the heck were they thinking?" FTP has many problems that the firewall administrator needs to deal with. What's worse, the problems the administrator must face are different between making ftp clients work and making ftp servers work.

Within the FTP protocol, there are two forms of data transfer, called active and passive. Active transfers are those where the server connects to an open port on the client to send data. Conversely, passive transfers are those where the client connects to the server to receive data.

8.2.1. Running an FTP Server

In running an FTP server, handling Active FTP sessions is easy to setup. At the same time, handling Passive FTP sessions is a big problem. First we'll cover how to handle Active FTP, then move on to Passive. Generally, we can handle Active FTP sessions like we would an incoming HTTP or SMTP connection; just open the ftp port and let `keep state` do the rest:

```
pass in quick proto tcp from any to 20.20.20.20/32 port = 21 flags S keep state
pass out proto tcp all keep state
```

These rules will allow Active FTP sessions, the most common type, to your ftp server on 20.20.20.20.

The next challenge becomes handling Passive FTP connections. Web browsers default to this mode, so it's becoming quite popular and as such it should be supported. The problem with passive connections are that for every passive connection, the server starts listening on a new port (usually above 1023). This is essentially like creating a new unknown service on the server. Assuming we have a good firewall with a default-deny policy, that new service will be blocked, and thus Active FTP sessions are broken. Don't despair! There's hope yet to be had.

A person's first inclination to solving this problem might be to just open up all ports above 1023. In truth, this will work:

```
pass in quick proto tcp from any to 20.20.20.20/32 port > 1023 flags S keep state
pass out proto tcp all keep state
```

This is somewhat unsatisfactory, though. By letting everything above 1023 in, we actually open ourselves up for a number of potential problems. While 1-1023 is the designated area for server services to run, numerous programs decided to use numbers higher than 1023, such as `nfsd` and `X`.

The good news is that your FTP server gets to decide which ports get assigned to active sessions. This means that instead of opening all ports above 1023, you can allocate ports 15001-19999 as ftp ports and only open that range of your firewall up. In `wu-ftpd`, this is done with the `passive ports` option in `ftppaccess`. Please see the man page on `ftppaccess` for details in `wu-ftpd` configuration. On the `ipfilter` side, all we need do is setup corresponding rules:

```
pass in quick proto tcp from any to 20.20.20.20/32 port 15000 >< 20000 flags S keep state
pass out proto tcp all keep state
```

If even this solution doesn't satisfy you, you can always hack IPF support into your FTP server, or FTP server support into IPF.

8.2.2. Running an FTP Client

While FTP server support is still less than perfect in IPF, FTP client support has been working well since 3.3.3. As with FTP servers, there are two types of ftp client transfers: passive and active.

The simplest type of client transfer from the firewall's standpoint is the passive transfer. Assuming you're keeping state on all outbound tcp sessions, passive transfers will work already. If you're not doing this already, please consider the following:

```
pass out proto tcp all keep state
```

The second type of client transfer, active, is a bit more troublesome, but nonetheless a solved problem. Active transfers cause the server to open up a second connection back to the client for data to flow through. This is normally a problem when there's a firewall in the middle, stopping outside connections from coming back in. To solve this, ipfilter includes an `ipnat` proxy which temporarily opens up a hole in the firewall just for the FTP server to get back to the client. Even if you're not using `ipnat` to do nat, the proxy is still effective. The following rules is the bare minimum to add to the `ipnat` configuration file (`ep0` should be the interface name of the outbound network connection):

```
map ep0 0/0 -> 0/32 proxy port 21 ftp/tcp
```

For more details on ipfilter's internal proxies, see section 3.6

8.3. Assorted Kernel Variables

There are some useful kernel tunes that either need to be set for ipf to function, or are just generally handy to know about for building firewalls. The first major one you must set is to enable IP Forwarding, otherwise ipf will do very little, as the underlying ip stack won't actually route packets.

IP Forwarding:

openbsd:

```
net.inet.ip.forwarding=1
```

freebsd:

```
net.inet.ip.forwarding=1
```

solaris:

```
ndd -set /dev/ip ip_forwarding 1
```

Ephemeral Port Adjustment:

openbsd:

```
net.inet.ip.portfirst = 25000
```

freebsd:

```
net.inet.ip.portrange.first = 25000 net.inet.ip.portrange.last = 49151
```

solaris:

```
ndd -set /dev/tcp tcp_smallest_anon_port 25000
```

```
ndd -set /dev/tcp tcp_largest_anon_port 65535
```

Other Useful Values:

openbsd:

```
net.inet.ip.sourceroute = 0
```

```
net.inet.ip.directed-broadcast = 0
```

freebsd:

```
net.inet.ip.sourceroute=0
```

```
net.ip.accept_sourceroute=0
```

solaris:

```
ndd -set /dev/ip ip_forward_directed_broadcasts 0
```

```
ndd -set /dev/ip ip_forward_src_routed 0
```

```
ndd -set /dev/ip ip_respond_to_echo_broadcast 0
```

In addition, freebsd has some ipf specific sysctl variables.

```
net.inet.ipf.fr_flags: 0
net.inet.ipf.fr_pass: 514
net.inet.ipf.fr_active: 0
net.inet.ipf.fr_tcpidletimeout: 864000
net.inet.ipf.fr_tcpclosewait: 60
net.inet.ipf.fr_tcplastack: 20
net.inet.ipf.fr_tcptimeout: 120
net.inet.ipf.fr_tcpclosed: 1
net.inet.ipf.fr_udptimeout: 120
net.inet.ipf.fr_icmptimeout: 120
net.inet.ipf.fr_defnatage: 1200
net.inet.ipf.fr_ipfrttl: 120
net.inet.ipf.ipl_unreach: 13
net.inet.ipf.ipl_initd: 1
net.inet.ipf.fr_authsize: 32
net.inet.ipf.fr_authused: 0
net.inet.ipf.fr_defaultauthage: 600
```

9. Fun with ipf!

This section doesn't necessarily teach you anything new about ipf, but it may raise an issue or two that you haven't yet thought up on your own, or tickle your brain in a way that you invent something interesting that we haven't thought of.

9.1. Localhost Filtering

A long time ago at a university far, far away, Weitse Venema created the tcp-wrapper package, and ever since, it's been used to add a layer of protection to network services all over the world. This is good. But, tcp-wrappers have flaws. For starters, they only protect TCP services, as the name suggests. Also, unless you run your service from inetd, or you have specifically compiled it with libwrap and the appropriate hooks, your service isn't protected. This leaves gigantic holes in your host security. We can plug these up by using ipf on the local host. For example, my laptop often gets plugged into or dialed into networks that I don't specifically trust, and so, I use the following rule set:

```
pass in quick on lo0 all
pass out quick on lo0 all

block in log all
block out all

pass in quick proto tcp from any to any port = 113 flags S keep state
pass in quick proto tcp from any to any port = 22 flags S keep state
pass in quick proto tcp from any port = 20 to any port 39999 >< 45000 flags S keep state

pass out quick proto icmp from any to any keep state
pass out quick proto tcp/udp from any to any keep state keep frags
```

It's been like that for quite a while, and I haven't suffered any pain or anguish as a result of having ipf loaded up all the time. If I wanted to tighten it up more, I could switch to using the NAT ftp proxy and I could add in some rules to prevent spoofing. But even as it stands now, this box is far more restrictive about what it presents to the local network and beyond than the typical host does. This is a good thing if you happen to run a machine that allows a lot of users on it, and you want to make sure one of them doesn't happen to start up a service they weren't supposed to. It won't stop a malicious hacker with root access from adjusting your ipf rules and starting a service anyway, but it will keep the "honest" folks honest, and your weird services safe, cozy and warm even on a malicious LAN. A big win, in my opinion. Using local host filtering in addition to a somewhat less-restrictive "main firewall" machine can solve many performance issues as well as *political* nightmares like "Why doesn't ICQ work?" and "Why can't I put a web server on my own workstation! It's MY WORKSTATION!!" Another very big win. Who says you can't have security and convenience at the same time?

9.2. What Firewall? Transparent filtering.

One major concern in setting up a firewall is the integrity of the firewall itself. Can somebody break into your firewall, thereby subverting its ruleset? This is a common problem administrators must face, particularly when they're using firewall solutions on top of their Unix/NT machines. Some use it as an argument for blackbox hardware solutions, under the flawed notion that inherent obscurity of their closed system increases their security. We have a better way.

Many network admins are familiar with the common ethernet bridge. This is a device that connects two separate ethernet segments to make them one. An ethernet bridge is typically used to connect separate buildings, switch network speeds, and extend maximum wire lengths. Hubs and switches are common bridges, sometimes they're just 2 ported devices called repeaters. Recent versions of Linux, OpenBSD, NetBSD, and FreeBSD include code to convert \$1000 PCs into \$10 bridges, too! What all bridges tend to have in common is that though they sit in the middle of a connection between two machines, the two machines don't know the bridge is there. Enter ipfilter and OpenBSD.

Ethernet bridging takes place at Layer2 on the ISO stack. IP takes place on Layer3. IP Filter is primarily concerned with Layer3, but dabbles in Layer2 by working with interfaces. By mixing IP filter with OpenBSD's bridge device, we can create a firewall that is both invisible and unreachable. The system needs no IP address, it doesn't even need to reveal its ethernet address. The only telltale sign that the filter might be there is that latency is somewhat higher than a piece of cat5 would normally make it, and that packets don't seem to make it to their final destination.

The setup for this sort of ruleset is surprisingly simple, too. In OpenBSD, the first bridge device is named `bridge0`. Say we have two ethernet cards in our machine as well, `xl0` and `xl1`. To turn this machine into a bridge, all one need do is enter the following three commands:

```
brconfig bridge0 add xl0 add xl1 up
ifconfig xl0 up
ifconfig xl1 up
```

At this point, all traffic arriving on `xl0` is sent out `xl1` and all traffic on `xl1` is sent out `xl0`. You'll note that neither interface has been assigned an IP address, nor do we need assign one. All things considered, it's likely best we not add one at all.

Rulesets behave essentially the as the always have. Though there is a `bridge0` interface, we don't filter based on it. Rules continue to be based upon the particular interface we're using, making it important which network cable is plugged into which network card in the back of the machine. Lets start with some basic filtering to illustrate what's happened. Assume the network used to look like this:

```
20.20.20.1 <-----> 20.20.20.0/24 network hub
```

That is, we have a router at 20.20.20.1 connected to the 20.20.20.0/24 network. All packets from the 20.20.20.0/24 network go through 20.20.20.1 to get to the outside world and vice versa. Now we add the IpF Bridge:

```
20.20.20.1 <-----/xl0 IpFBridge xl1/-----> 20.20.20.0/24 network hub
```

We also have the following ruleset loaded on the IpFBridge host:

```
pass in quick all
pass out quick all
```

With this ruleset loaded, the network is functionally identical. As far as the 20.20.20.1 router is concerned, and as far as the 20.20.20.0/24 hosts are concerned, the two network diagrams are identical. Now lets change the ruleset some:

```
block in quick on xl0 proto icmp
pass in quick all
pass out quick all
```

Still, 20.20.20.1 and 20.20.20.0/24 think the network is identical, but if 20.20.20.1 attempts to ping 20.20.20.2, it will never get a reply. What's more, 20.20.20.2 won't even get the packet in the first place. IPfilter will intercept the packet before it even gets to the other end of the virtual wire. We can put a

bridged filter anywhere. Using this method we can shrink the network trust circle down an individual host level (given enough ethernet cards:-)

Blocking icmp from the world seems kind of silly, especially if you're a sysadmin and like pinging the world, to traceroute, or to resize your MTU. Lets construct a better ruleset and take advantage of the original key feature of ipf: stateful inspection.

```
pass in quick on xl1 proto tcp keep state
pass in quick on xl1 proto udp keep state
pass in quick on xl1 proto icmp keep state
block in quick on xl0
```

In this situation, the 20.20.20.0/24 network (perhaps more aptly called the xl1 network) can now reach the outside world, but the outside world can't reach it, and it can't figure out why, either. The router is accessible, the hosts are active, but the outside world just can't get in. Even if the router itself were compromised, the firewall would still be active and successful.

So far, we've been filtering by interface and protocol only. Even though bridging is concerned layer2, we can still discriminate based on IP address. Normally we have a few services running, so our ruleset may look like this:

```
pass in quick on xl1 proto tcp keep state
pass in quick on xl1 proto udp keep state
pass in quick on xl1 proto icmp keep state
block in quick on xl1 # nuh-uh, we're only passing tcp/udp/icmp sir.
pass in quick on xl0 proto udp from any to 20.20.20.2/32 port=53 keep state
pass in quick on xl0 proto tcp from any to 20.20.20.2/32 port=53 flags S keep state
pass in quick on xl0 proto tcp from any to 20.20.20.3/32 port=25 flags S keep state
pass in quick on xl0 proto tcp from any to 20.20.20.7/32 port=80 flags S keep state
block in quick on xl0
```

Now we have a network where 20.20.20.2 is a zone serving name server, 20.20.20.3 is an incoming mail server, and 20.20.20.7 is a web server.

Bridged IP Filter is not yet perfect, we must confess.

First, You'll note that all the rules are setup using the `in` direction instead of a combination of `in` and `out`. This is because the `out` direction is presently unimplimented with bridging in OpenBSD. This was originally done to prevent vast performance drops using multiple interfaces. Work has been done in speeding it up, but it remains unimplimented. If you really want this feature, you might try your hand at working on the code or asking the OpenBSD people how you can help.

Second, using IP Filter with bridging makes the use of IPF's NAT features inadvisable, if not downright dangerous. The first problem is that it would give away that there's a filtering bridge. The second problem would be that the bridge has no IP address to masquerade with, which will most assuredly lead to confusion and perhaps a kernel panic to boot. You can, of course, put an IP address on the outbound interface to make NAT work, but part of the glee of bridging is thus diminished.

9.2.1. Using Transparent Filtering to Fix Network Design Mistakes

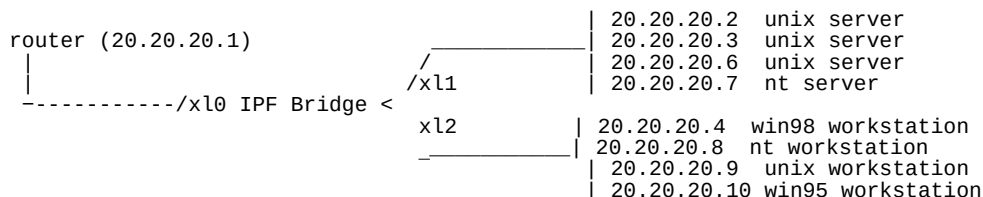
Many organizations started using IP well before they thought a firewall or a subnet would be a good idea. Now they have class-C sized networks or larger that include all their servers, their workstations, their routers, coffee makers, everything. The horror! Renumbrering with proper subnets, trust levels, filters, and so are in both time consuming and expensive. The expense in hardware and man hours alone is enough to make most organizations unwilling to really solve the problem, not to mention the downtime involved. The typical problem network looks like this:

20.20.20.1	router	20.20.20.6	unix server
20.20.20.2	unix server	20.20.20.7	nt workstation
20.20.20.3	unix server	20.20.20.8	nt server
20.20.20.4	win98 workstation	20.20.20.9	unix workstation
20.20.20.5	intelligent switch	20.20.20.10	win95 workstation

Only it's about 20 times larger and messier and frequently undocumented. Ideally, you'd have all the trust-ing servers in one subnet, all the work- stations in another, and the network switches in a third. Then the router would filter packets between the subnets, giving the workstations limited access to the servers, nothing access to the switches, and only the sysadmin's workstation access to the coffee pot. I've never seen a

class-C sized network with such coherency. IP Filter can help.

To start with, we're going to separate the router, the workstations, and the servers. To do this we're going to need 2 hubs (or switches) which we probably already have, and an IPF machine with 3 ethernet cards. We're going to put all the servers on one hub and all the workstations on the other. Normally we'd then connect the hubs to each other, then to the router. Instead, we're going to plug the router into IPF's x10 interface, the servers into IPF's x11 interface, and the workstations into IPF's x12 interface. Our network diagram looks something like this:



Where once there was nothing but interconnecting wires, now there's a filtering bridge that not a single host needs to be modified to take advantage of. Presumably we've already enabled bridging so the network is behaving perfectly normally. Further, we're starting off with a ruleset much like our last ruleset:

```

pass in quick on x10 proto udp from any to 20.20.20.2/32 port=53 keep state
pass in quick on x10 proto tcp from any to 20.20.20.2/32 port=53 flags S keep state
pass in quick on x10 proto tcp from any to 20.20.20.3/32 port=25 flags S keep state
pass in quick on x10 proto tcp from any to 20.20.20.7/32 port=80 flags S keep state
block in quick on x10
pass in quick on x11 proto tcp keep state
pass in quick on x11 proto udp keep state
pass in quick on x11 proto icmp keep state
block in quick on x11 # nuh-uh, we're only passing tcp/udp/icmp sir.
pass in quick on x12 proto tcp keep state
pass in quick on x12 proto udp keep state
pass in quick on x12 proto icmp keep state
block in quick on x12 # nuh-uh, we're only passing tcp/udp/icmp sir.

```

Once again, traffic coming from the router is restricted to DNS, SMTP, and HTTP. At the moment, the servers and the workstations can exchange traffic freely. Depending on what kind of organization you are, there might be something about this network dynamic you don't like. Perhaps you don't want your workstations getting access to your servers at all? Take the x12 ruleset of:

```

pass in quick on x12 proto tcp keep state
pass in quick on x12 proto udp keep state
pass in quick on x12 proto icmp keep state
block in quick on x12 # nuh-uh, we're only passing tcp/udp/icmp sir.

```

And change it to:

```

block in quick on x12 from any to 20.20.20.0/24
pass in quick on x12 proto tcp keep state
pass in quick on x12 proto udp keep state
pass in quick on x12 proto icmp keep state
block in quick on x12 # nuh-uh, we're only passing tcp/udp/icmp sir.

```

Perhaps you want them to just get to the servers to get and send their mail with IMAP? Easily done:

```

pass in quick on x12 proto tcp from any to 20.20.20.3/32 port=25
pass in quick on x12 proto tcp from any to 20.20.20.3/32 port=143
block in quick on x12 from any to 20.20.20.0/24
pass in quick on x12 proto tcp keep state
pass in quick on x12 proto udp keep state
pass in quick on x12 proto icmp keep state
block in quick on x12 # nuh-uh, we're only passing tcp/udp/icmp sir.

```

Now your workstations and servers are protected from the outside world, and the servers are protected from your workstations.

Perhaps the opposite is true, maybe you want your workstations to be able to get to the servers, but not the outside world. After all, the next generation of exploits is breaking the clients, not the servers. In this case, you'd change the x12 rules to look more like this:

```
pass in quick on xl2 from any to 20.20.20.0/24
block in quick on xl2
```

Now the servers have free reign, but the clients can only connect to the servers. We might want to batten down the hatches on the servers, too:

```
pass in quick on xl1 from any to 20.20.20.0/24
block in quick on xl1
```

With the combination of these two, the clients and servers can talk to each other, but neither can access the outside world (though the outside world can get to the few services from earlier). The whole ruleset would look something like this:

```
pass in quick on xl0 proto udp from any to 20.20.20.2/32 port=53 keep state
pass in quick on xl0 proto tcp from any to 20.20.20.2/32 port=53
pass in quick on xl0 proto tcp from any to 20.20.20.3/32 port=25
pass in quick on xl0 proto tcp from any to 20.20.20.7/32 port=80
block in quick on xl0
pass in quick on xl1 from any to 20.20.20.0/24
block in quick on xl1
pass in quick on xl2 from any to 20.20.20.0/24
block in quick on xl2
```

So remember, when your network is a mess of twisty IP addresses and machine classes, transparent filtered bridges can solve a problem that would otherwise be lived with and perhaps someday exploited.

9.3. Drop-Safe Logging With `dup-to` and `to`.

Until now, we've been using the filter to drop packets. Instead of dropping them, let's consider passing them on to another system that can do something useful with this information beyond the logging we can perform with `ipmon`. Our firewall system, be it a bridge or a router, can have as many interfaces as we can cram into the system. We can use this information to create a "drop-safe" for our packets. A good example of a use for this would be to implement an intrusion detection network. For starters, it might be desirable to hide the presence of our intrusion detection systems from our real network so that we can keep them from being detected.

Before we get started, there are some operational characteristics that we need to make note of. If we are only going to deal with blocked packets, we can use either the `to` keyword or the `fastroute` keyword. (We'll cover the differences between these two later) If we're going to pass the packets like we normally would, we need to make a copy of the packet for our drop-safe log with the `dup-to` keyword.

9.3.1. The `dup-to` Method

If, for example, we wanted to send a copy of everything going out the `xl3` interface off to our drop-safe network on `ed0`, we would use this rule in our filter list:

```
pass out on xl3 dup-to ed0 from any to any
```

You might also have a need to send the packet directly to a specific IP address on your drop-safe network instead of just making a copy of the packet out there and hoping for the best. To do this, we modify our rule slightly:

```
pass out on xl3 dup-to ed0:192.168.254.2 from any to any
```

But be warned that this method will alter the copied packet's destination address, and may thus destroy the usefulness of the log. For this reason, we recommend only using the known address method of logging when you can be certain that the address that you're logging to corresponds in some way to what you're logging for (e.g.: don't use "192.168.254.2" for logging for both your web server and your mail server, since you'll have a hard time later trying to figure out which system was the target of a specific set of packets.)

This technique can be used quite effectively if you treat an *IP Address* on your drop-safe network in much the same way that you would treat a *Multicast Group* on the real internet. (e.g.: "192.168.254.2" could be the channel for your http traffic analysis system, "23.23.23.23" could be your channel for telnet sessions, and so on.) You don't even need to actually have this address set as an address or alias on any of your analysis systems. Normally, your `ipfilter` machine would need to ARP for the new destination address

(using `dup-to ed0:192.168.254.2` style, of course) but we can avoid that issue by creating a static arp entry for this "*channel*" on our ipfilter system.

In general, though, `dup-to ed0` is all that is required to get a new copy of the packet over to our drop-safe network for logging and examination.

9.3.2. The to Method

The `dup-to` method does have an immediate drawback, though. Since it has to make a copy of the packet and optionally modify it for its new destination, it's going to take a while to complete all this work and be ready to deal with the next packet coming in to the ipfilter system.

If we don't care about passing the packet to its normal destination and we were going to block it anyway, we can just use the `to` keyword to push this packet past the normal routing table and force it to go out a different interface than it would normally go out.

```
block in quick on xl0 to ed0 proto tcp from any to any port < 1024
```

we use `block quick` for `to` interface routing, because like `fastroute`, the `to` interface code will generate two packet paths through ipfilter when used with `pass`, and likely cause your system to panic.

<https://web.archive.org/web/20030831153359/http://cnscenter.future.co.kr:80/resource/rsc-center/presentation/intel/fall2002/VOL051PS.pdf>

Platform Management IPMI Controllers, Sensors, and Tools

Tom Slaight

**Server Management Architect
Enterprise Platforms Group
Intel Corporation**

September 11, 2002

Introduction

- **Audience:**
**Architects, Technical Managers,
Firmware Leads, and Hardware
Designers**
 - Involved in architecture, component selection, debug, test, or design of server platform and peripheral management subsystems
- **Focus: IPMI-based implementations**
 - Hardware components
 - Hardware and firmware development tools

Disclaimers

- **A good starting point**
 - ... but not a comprehensive list of vendors or available technology
 - Listing of particular vendors and products does not constitute an endorsement by Intel or the IPMI Promoters
- **No guarantees on accuracy of information provided**
 - Contact vendors directly for complete specifications and availability information

“Architect’s Pick”

Agenda

- **IPMI Update & Architecture Overview**
- **Baseboard Management & Enclosure/Peripheral Controllers**
- **Sensor Devices**
- **Putting it all together - design advice and tools**
- **Summary**



IPMI

Intelligent Platform Management Interface

- Defines a standardized, abstracted, message-based interface to intelligent platform management hardware
- Defines standardized records for describing platform management devices and their characteristics

Promoters:



Adopters: Over 125 and growing

<http://developer.intel.com/design/servers/ipmi>



Initiative News

Updated Errata (e.o. Sept), Conformance Test Suite and 64- & 32-bit .NET/Windows* 2000 Drivers Available

intel.

United States Home | Select a Location

Products Support

Home Computing Business Developer Reseller / Provider

Search

Advanced Search

Developer

Hardware Design

Software Development

Download Software/Drivers

Design Support/Services

Events, Training & Pubs

R&D/Initiatives

server home

Server Building Blocks

Itanium™ Processor Family

Hardware Developer's Resource Center

Software Developer's Area

Industry Technology Specifications

Community

Related

Intel

Resources

Intelligent Platform Management

New revision (8/05/02):

driver for IA-64 and IA-32 under Windows* .NET/2000 OS

driver implementation for IA-64 and IA-32 under .NET/2000 OS available to IPMI adopters only.

IPMI 1.5 Conformance Test Suite (ICTS) Prototype 5.02

includes IPMI v1.0 and IPMI v1.5 automated conformance tests, IPMI v1.5 CMDTOOL for manual IPMI v1.5 testing, support for PCI* card based IPMB and SMBus testing, and support for IPMI v1.5 new interfaces including LAN, Serial and SMBus. ICTS 5.02 is an update to ICTS 5.01 and adds new tests for IPMI 1.5 commands and includes

developer.intel.com/design/servers/ipmi

intel.

Intel Developer Forum

Lenovo

Ex. 1039 - Page 71

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

Other Initiative News

- **ASF 2.0**
 - **A**lert **S**tandard **F**ormat specification from DMTF
 - Adds authenticated remote power and reset control over ASF 1.0
 - Near finalization
- **PICMG 3.x / AdvancedTCA***
 - IPMI additions for PICMG 3.x support
 - Network function for “AdvancedTCA” commands
 - Slot/Connector type for “AdvancedTCA” boards

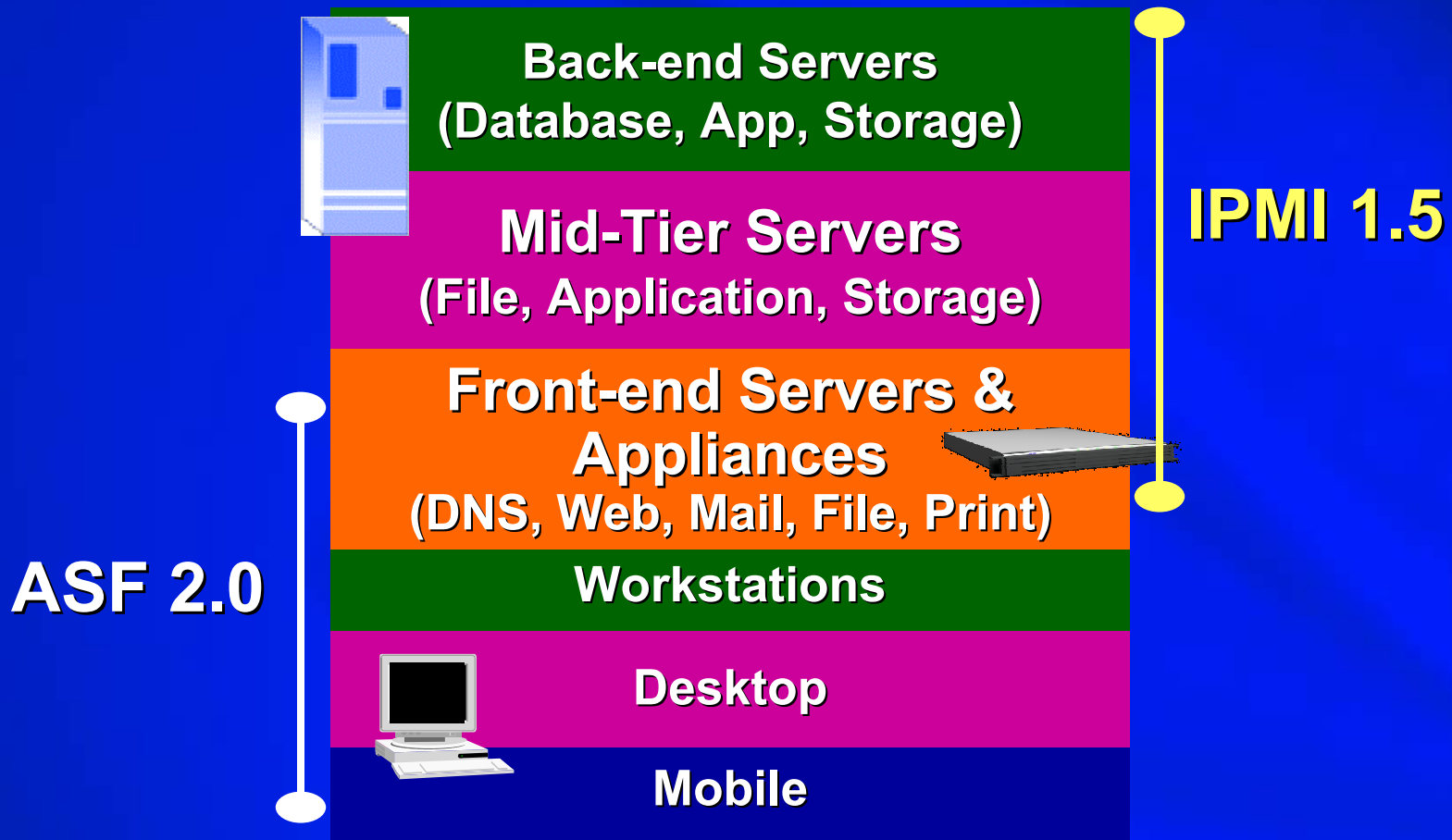
IPMI evolves with new management standards

Platform Management Technologies

IPMI 1.5 ASF 2.0	System Health & Security Alerts	• Health Alerts (Temperature, Voltage, Fan, etc. -- 128 definable alerts) • Security Alerts (Chassis Intrusion, LAN Heartbeat, System Password Violation) • BIOS Messages & Alerts • OS Hung Watchdog Timer
	Authenticated Remote Control**	• Processor Missing (startup Watchdog) • Power up/down/cycle/reset • Boot & Boot Path Options
	Status Info	• System State • System ASF Capabilities • Presence Ping/Pong
	Monitoring	• Abstracted Local and Remote Monitoring • Health Sensor "Present Reading" Access (Temperature, Voltage, Fan, etc.)
	Inventory and Logging	• FRU/Inventory Data Access • Event Logging
	Extended Out-of-Band Access	• Serial/Modem Access • Text-based Access • Inter-Chassis Access • Multi-level, Multi-user Security
	Extended Alerting	• Alerts to multiple destinations • Paging via Modem
	Automatic Actions	• Platform Event Filtering (PEF) - Configurable, Event-based automatic recovery & alerts
	Scalability / Extensibility	• Utilizes independent BMC • "Unlimited" Events & Sensors • Extensible Sensor/Event Busses

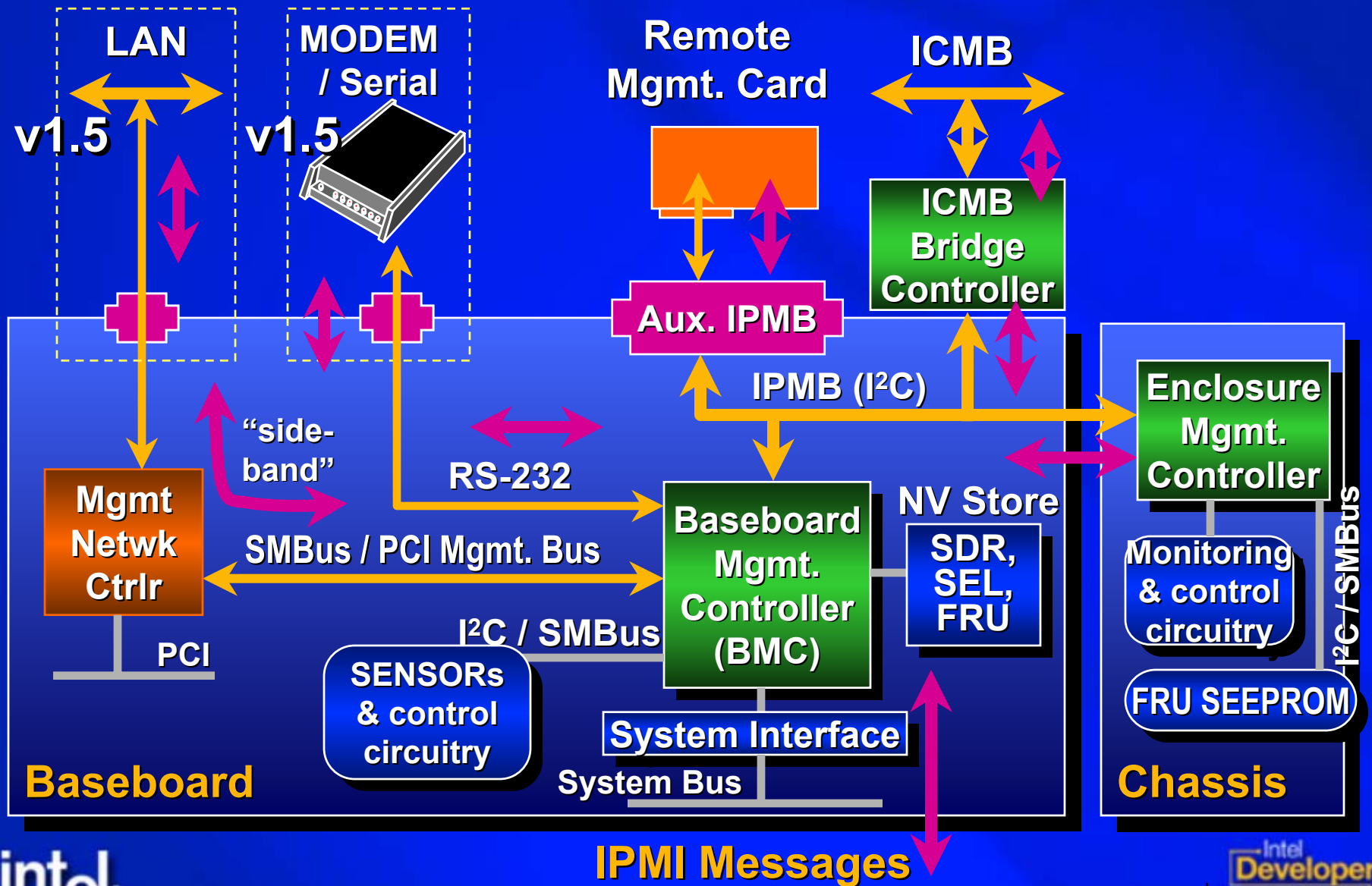
** IPMI supports multi-user as well as multilevel authentication

ASF/IPMI Typical Applications



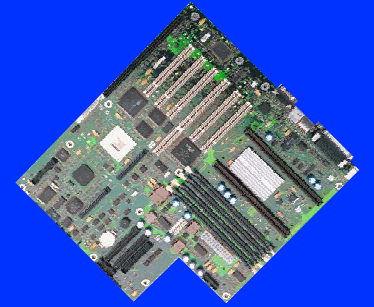
**IPMI and ASF are complementary
and cooperating technologies**

IPMI v1.5 Architecture



Agenda

- **IPMI Update & Architecture Overview**
- **Baseboard Management & Enclosure/Peripheral Controllers**
- **Sensor Devices**
- **Putting it all together – design advice and tools**
- **Summary**



System Interface

- **Built-in System Interfaces**
 - For IPMI, three types possible:
 - KCS (Keyboard Controller Style) most popular.
 - BT (Block Transfer) is fastest.
 - SMIC (Server Mgmt. Interface Chip) for implementation via external ASIC or FPGA [not recommended for new designs]
- **“Low glue” connection to chip set**
 - IPMI specifications support **memory mapped implementations** as well as original I/O mapped
- **System Interface interrupt support**
- **Multiple Built-in System Interfaces**
 - KCS for ACPI EC (Embedded Controller) interface
 - Additional interface can support OEM differentiation and/or SMI Handler access

I²C/SMBus Support

- **Recommend three Master-Slave interfaces for general purpose server BMC**
 - For IPMB, PCI SMBus, and LAN/future
- **Interfaces support multiple slave addresses**
 - One fixed for I²C broadcast (00h) for IPMB
 - One programmable for device as IPMI target
 - Second programmable for snoop/SMBus host target
- **Multiple private management bus support**
 - Reduces need for external I²C bus multiplexing
 - Can be 'bit banded' using I/O pins for simple sensor devices
- **SMBus and I²C compatibility**
 - For PCI SMBus and IPMB support and sensor device flexibility

Flexible I/O

- **Selectable open-drain or driven outputs**
 - Reduces need for external pullups
 - Can implement 'bit banded' private I²C/SMBus
- **3.3V Standby operation with 5V tolerance**
 - IPMB, SSI power supply FRU and other signals still 5V
- **High current outputs (>12 mA)**
 - For direct LED drive
- **Multiple External Interrupts**
 - For asynchronous event capture

UARTs and Fan Monitoring/Control

- **Multiple UARTs with hardware handshake support**
 - For ICMB & IPMI-over-Modem support
 - For redirection over LAN features
 - To support redundant connections for modular server designs
- **Timers or Counters for Tach FAN Speed monitoring**
- **Pulse-width Modulator (PWM) or D/A outputs for FAN Speed control**

Built-in Analog-to-Digital

- **Need for at least 7 voltages is common**
 - E.g. Processor 1 & 2, 3.3V, 3.3Vaux, 5V, 12V, -12V
- **Many server systems require more**
 - >2 processors, SCSI terminations, bus, cache/chipset, etc.
 - 19 or more is not uncommon!
- **Conversion rate usually not an issue...**
 - A single converter with analog multiplexing works well
- **But accuracy, tolerance, and resolution are**
 - Recommend at least 8-bit resolution, +/-1 bit tolerance, and +/-1% accuracy

Low accuracy paid for with system margins

Firmware support

- **Development and Debug Support**
 - Vendor-provided debug tools
 - Test port or emulator support
 - Commercial high-level language & code debugging tools
 - Commercial RTOS options
- **IPMI Firmware / SDK**
 - Typical SDKs support customer-developed extensions for 'Value added' features
 - Vendors may also offer customization services

Firmware and SDKs yield faster TTM for IPMI-based designs

Extensibility, Headroom, and Low Cost

- **Low Power Consumption**
 - <50 mA is a pretty good target
- **Scalability**
 - Can one controller fit multiple products?
 - ...Or is it part of a 'family' that can?
- **ROM/RAM Upgrade Path**
- **Performance Headroom**
 - Authentication & Encryption for potential future interfaces, e.g. Web, demands compute power
- **External Expansion Capability**
- **Package and Board Space**
 - BGA packaging can provide higher density
 - Watch tradeoffs between package size & overall solution cost
- **Low Unit Cost**

Management Controllers*

= new since last presentation

Mfr.	product	core	system I/F	I ² C	A/D	serial	type	App	LED drv	special features	IPMI F/W	Avail.
Agilent	eRMC	Contact vendor...										
Dallas Semi.	DS80CH11	8032 compat.	3 KCS via ISA	2 m/s	Y	1	8051	BMC			no	now
Hitachi	H8/3337Y	H8 8-bit	1 KCS via ISA	1 m/s	Y	Y	SCI	BMC		D/A, PWM	no	now
	H8S/2148	H8S 16-bit	4 KCS via ISA	2 m/s	Y	Y	SCI	BMC		D/A, PWM	no	now
National Semi.	87431	mini BMC										
Philips	80C652	8032 compat.	external req'd	1 m/s	no	1	8051	Sat / Bridge			no	now
	80C552	8032 compat.	external req'd	1 m/s	8 ch	1	8051	Sat / BMC			no	now
Qlogic	Zircon BL	ARM7/ TDMI	2 KCS via LPC	2 m/s	6**	1	"16550"	BMC	yes	2 PWM, 2 fan tach, USP,	yes	4Q02
Qlogic	Zircon UL	ARM7/ TDMI	2 KCS via LPC	2 m/s	6**	1	"16550"	BMC	yes	3 PWM, 4 fan tach, USP,	yes	4Q02

** Comparators

... Continued next page

Management Controllers*

Mfr.	product	core	system I/F	I ² C	A/D	serial	type	App	LED drv	special features	IPMI F/W	Avail.
Qlogic	Zircon CP 128 QFP	ARM7/ TDMI	BT, 2 KCS LPC/ISA	2 m/s	6 ch	1	same	BMC / cPCI	yes	2 PWM, 4 fan tach	yes	now
Qlogic	Zircon	ARM7/ TDMI	3 KCS via LPC, ISA	3 m/s	10 ch	2	"16550"	BMC	yes	8 PWM, ICMB assist, 12 fan tach	yes	now
Qlogic	Zircon Lite 160 PQFP	ARM7/ TDMI	BT, 2 KCS LPC/ISA	2 m/s	8 ch	1	same	BMC / cPCI	yes	2 PWM, 4 fan tach	yes	now
Vitesse Semi.	VSC210	R3000	3 KCS/ SMIC/ BT via LPC	3 m/s	no	3	ICE/GP FIFO'd ICMB	BMC / Sat	12mA	fan tach, ICMB assist	yes	now
Vitesse Semi.	VSC215	R3000	3 KCS/ SMIC/ BT via LPC	4 m/s	Y	4	ICE/GP FIFO'd ICMB	BMC / Sat	12mA	fan tach, ICMB assist	yes	now
Winbond	W83910F	8032 compat.	3 KCS/ SMIC/ BT via LPC	5 m/s	7	2	"16550"	BMC	yes	PWM, LCD Module I/F, 4 temp diode	yes	now

BMC availability expanding to match server segments



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

Baseboard Management Controllers

Qlogic* Zircon UL

- Target applications: 1P/2P Servers
- 32 GPIO (max), 4 Fan tach, 3 PWM
- 128-pin PQFP
- Virtual Storage Interface – supports ‘virtual floppy’
- Universal Serial Interface
 - for serial redirection / headless
- Samples: now, Production: 4Q02

Qlogic Zircon BL

- Target applications: server blades, high-end workstations
- 18 GPIO (max), 2 fan tach, 2 PWM
- 100-pin PQFP
- Virtual Storage Interface
- Universal Serial Interface
- Samples: now, Production: 4Q02



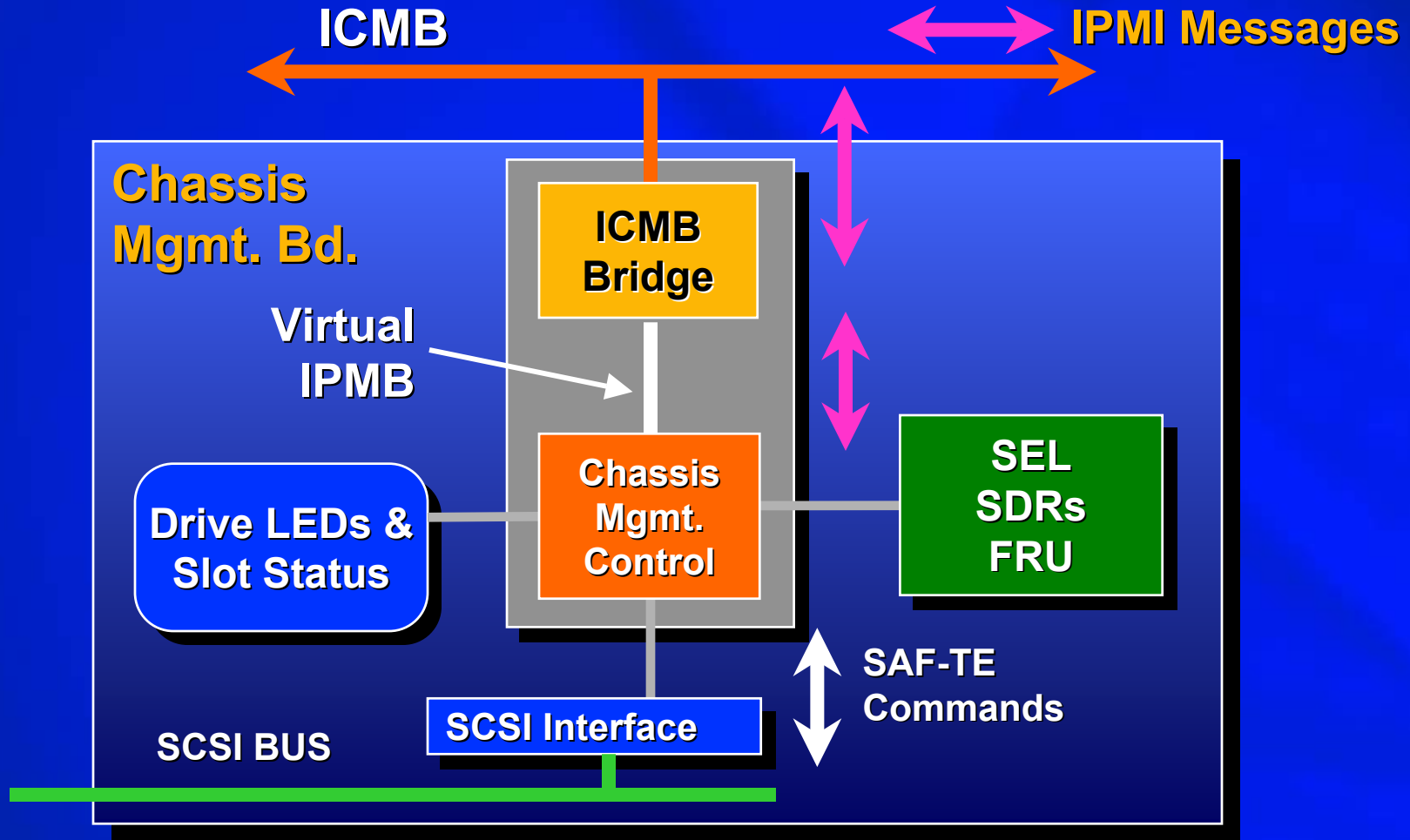
* Other names and brands may be claimed as the property of others.

Baseboard Management Controllers

National Semiconductor* PC87431M “mini-BMC”

- **Targets IPMI-based LAN remote mgmt.**
 - supports monitoring by local mgmt. s/w via SMBus
 - configurable polling of sensors similar to ASF
- **Authenticated IPMI LAN support for:**
 - System reset, SMI/NMI, and power control
 - settable ‘Boot Options’
 - FRU, System Event Log, and SDR access
 - Alerting via IPMI/PET SNMP Traps
- **“PEF-like” configurable actions on events**
 - power control, reset, fault light, NMI/SMI, and alert
- **Internal FLASH, RAM, NVRAM**
 - up to 512 bytes NV available for OEM use

Example Block Diagram



- SAF-TE = SCSI Accessed Fault-Tolerant Enclosures
- SES = ANSI SCSI Enclosure Services

Features to look for...

- **SAF-TE and SES Firmware**
 - Extensible / customizable?
 - Configurable LED definitions?
- **IPMI Firmware / SDK**
 - IPMB Support?
- **Support for external sensors or FRU devices**
 - e.g. FAN & power monitoring

**Firmware and SDKs yield faster TTM
for Enclosure/Peripheral Controllers**

Agenda

- IPMI Update & Architecture Overview
- Baseboard Management & Enclosure/Peripheral Controllers
- Sensor Devices
- Putting it all together – design advice and tools
- Summary



Sensors and Monitors with I²C/SMBus

- **Board Temperature Sensors**
 - Temperature sensor in package
 - May include digital outputs for fan control
- **Processor Temperature Sensors**
 - Thermal diode monitors plus built-in temperature
 - May include automatic fan control
- **Hardware Monitors**
 - Voltages for board and processors
 - Processor VID (voltage ID) monitor
 - Temperatures
 - built-in plus remote thermal diodes for processors
 - Fan speeds
 - Digital I/O or PWM for fan speed control

Focus on dual-processor monitoring

Sensors and Monitors with I²C/SMBus

- **Other composite sensor devices**
 - Focus on specific areas
 - Disk drive enclosure monitoring
 - Fan monitoring & control and voltage monitoring for processors
 - Fan and voltage monitoring for >2-way systems
 - See examples in backup slides
- **Sensor Trend: Automatic Fan Control**
 - Noise Reduction becomes more difficult for pedestal servers
 - Need for more granular / gradual speed control
 - Audible FAN Speed Cycling disconcerting to user
 - Need for per-fan or per-zone control
 - Driving all fans to same speed produces higher than needed noise levels

Agenda

- IPMI Update & Architecture Overview
- Baseboard Management & Enclosure/Peripheral Controllers
- Sensor Devices
- Putting it all together – design advice and tools
- Summary

Design Pointers

- **SMBus and I²C not directly compatible**
 - Timing and Electricals are close, but not identical
 - But masters and slave devices can be designed to work with both
 - And I²C and SMBus slave devices can typically be used on same bus
- **Slave Devices: Only use devices with data integrity checks on IPMB and PCI SMBus**
 - Place other devices on 'Private Management Busses' behind management controllers or other integrity-checked device
- **Masters: Include support for clearing the 'Stuck 0' condition**

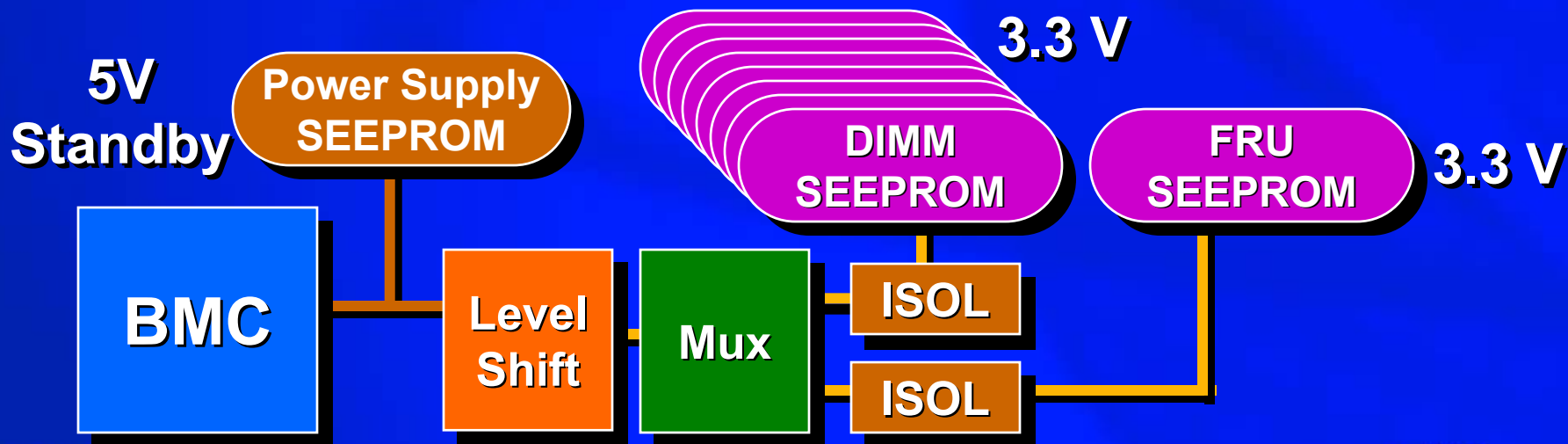
Design for SMBus and I²C compatibility

See IPMI Web Site for more info

I²C/SMBus Application

Bus Driving

- **Multiplexing** required when dealing with address option shortages or conflicts
- **Conversion** required when dealing with different bus voltage levels
- **Isolation** required when unpowered devices would 'short out' bus
- **Repeater** functionality when driving long busses or many loads (e.g. PCI SMBus)



I²C/SMBus Application

Bus Driving

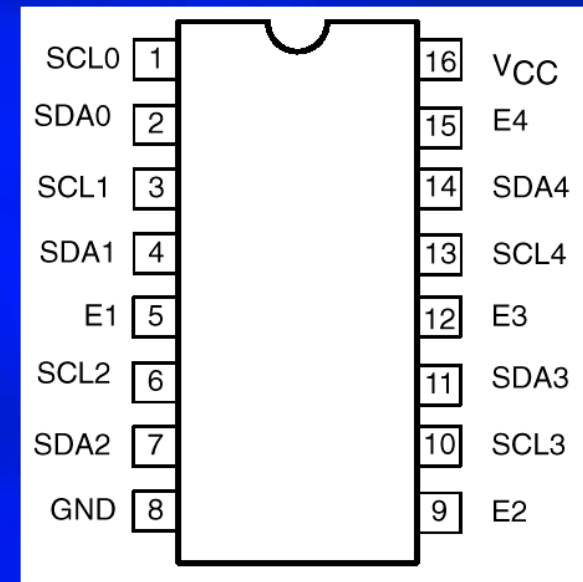
Philips Semiconductor* PCA9516 “5 channel I²C Hub”

- Allows *multiple* additional 400 pF bus segments
- Segments can be individually isolated
- Supports level translation (5V tolerant pins)
- Pins “Hi-Z” when device un-powered.
- Supports multi-master arbitration across the repeater

Possible applications:

Supporting PCI Management Bus on >8 PCI Slots, isolating SMBus to ‘hot-plug PCI’ slots, driving I²C to multiple system boards

PCA9515 single segment I²C Bus Repeater
also available



Testers & Monitors

- **Generators and monitors available**
 - Standalone and PC Hosted
 - Ready to run software
 - Software libraries and SDKs enable IPMI message parsing
- **Some tool vendors our developers have used**
 - Microcomputer Control Corporation
<http://www.mcc-us.com/>
 - Calibre UK Ltd
<http://www.calibreuk.com/>
 - Card also supported in IPMI conformance test suite
 - Telos EDV Systementwicklung GmbH
<http://www.lucit.de/>

IPMI Messaging Testing

- **IPMITOOL**
 - simple messaging utility available from IPMI Web Site
- **Debug tools from component vendors**
- **Software from I²C tester vendors**
 - use SDKs to extend tool for IPMI message parsing
- **IPMI Conformance Test**

IPMI Conformance Test

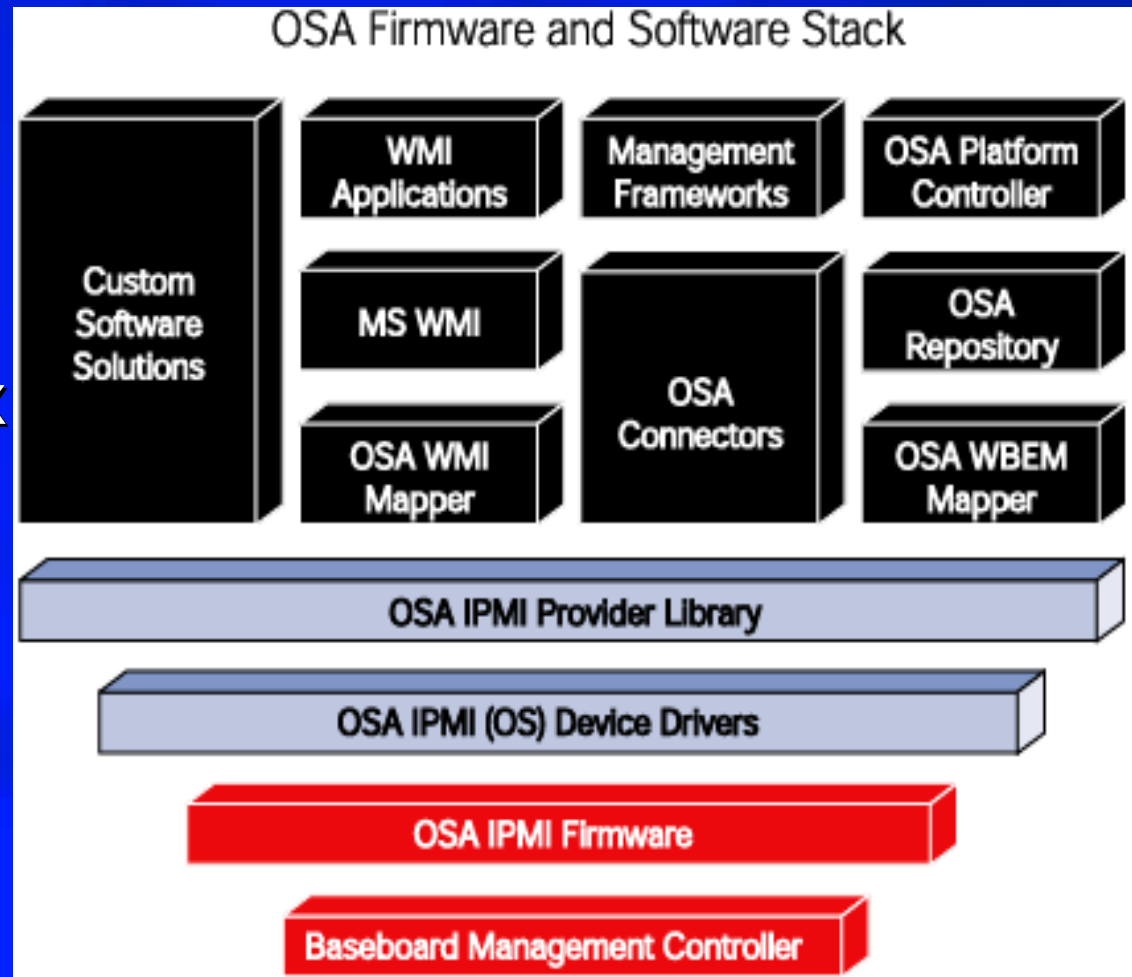
- **Automated testing for IPMI v1.5 & v1.0**
 - generates command conformance reports
- **Performs System interface accessible testing**
 - KCS and SMIC interfaces, Watchdog Timer, BMC Sensors
 - System event log (SEL), sensor data record (SDR), and FRU access
 - IPMB, ICMB testing
- **KCS protocol test**
- **IPMI v1.5 LAN, Serial, IPMB, and PCI SMBus test support**
- **Platform Event Filtering (PEF) and alerting**
- **Also usable as debug and development aid**
 - TCL script-based testing is user extensible
 - Examples provided

Speeds development and conformance testing

Software and Firmware building blocks

OSA Technologies

- Platform Mgmt. S/W and F/W for IPMI & PICMG 3.x
- Modular Architecture
 - Extensible
 - Scaleable across product lines



Agenda

- IPMI Update & Architecture Overview
- Baseboard Management & Enclosure/Peripheral Controllers
- Sensor Devices
- Putting it all together - design advice and tools
- Summary



Summary

- **New initiatives and specifications enable competitive server management features**
- **Management controllers and sensor devices available to fit your design and feature differentiation needs**
- **Tests & tools reduce system cost and design effort**
- **Help available today...**

Component & Tool Vendors*

- **Agilent Technologies**
contact.tm.agilent.com/tmo/datasheets/English/PRMC_ERMC.html, Niki Haines, niki_haines@agilent.com
- **Analog Devices, Inc.**
www.analog.com/pc & /temp-sensors, Paul Errico - paul.errico@analog.com
- **Dallas Semiconductor**
www.dalsemi.com
- **Hitachi Semiconductor (America), Inc.**
semiconductor.hitachi.com
- **Micro Computer Control Corporation**
www.mcc-us.com, Ed Thompson, ed.thompson@mcc-us.com
- **National Semiconductor Corporation**
www.national.com/appinfo/tempsensors/ & /advancedio, Hezi Friedman, hezi.friedman@nsc.com
- **Philips Semiconductors**
www.philipslogic.com/products/i2c, Joe Kochanski, joseph.kochanski@phlips.com
- **OSA Technologies**
www.osatechnologies.com, Steve Rokov, steve.rokov@osatechnologies.com
- **Qlogic Corporation**
www.qlogic.com, Mark Byrne-Quinn, mark.byrne-quinn@qlogic.com
- **Texas Instruments**
www.ti.com
- **Vitesse Semiconductor Corporation**
www.vitesse.com, Tom Brokaw, brokaw@vitesse.com
- **Winbond Electronics Corporation**
www.winbond.com.tw, Chad M.C. Wu, mcwu0@winbond.com.tw

Organizations / Specifications

- **IPMI (Intelligent Platform Management Interface) Specifications**
 - <http://developer.intel.com/design/servers/ipmi>
- **SMBus Specifications**
 - System Management Bus (SMBus) Specification, Version 2.0, August 3, 2000
<http://www.smbus.org>
- **I²C Specifications**
 - The I²C-BUS Specification, Version 2.1, January 2000
<http://semiconductors.philips.com/>
- **DMTF Pre-OS Working Group / ASF Specifications***
 - <http://www.dmtf.org>
- **PICMG / AdvancedTCA – PCI Industrial Computers Working Group**
 - <http://www.picmg.org>

IPMI Promoters, Adopters and Contributors

- ABIT Computer Corp.
- Acer Inc.
- Advanced Micro Devices, Inc.
- Agilent Technologies GmbH
- Alberta Microelectronics
- Allion Computer Inc.
- American Megatrends Inc.
- Arima Computer Corp.
- ASIS LTD.
- ASUSTek Computer, Inc.
- Avian Communications
- Axil Computer, Inc.
- Blue Wave Systems
- Bull S.A.
- Celestica
- C&D Technologies, Inc.
- ColoWATCH, Inc.
- Communication Automation Corporation
- Concurrent Technologies PLC
- CyberGuard Corporation
- Cyclades Corporation
- Data General Corporation
- Decru, Inc.
- Dell Computer Corporation
- Egenera, Inc.
- ElanVital Corporation
- Ericsson UAB
- Evans & Sutherland
- Eversys Corporation
- Exabyte Corporation
- First International Computer, Inc.
- Flextel SpA
- FORCE Computers GmbH
- Freedom Technologies Corporation
- Fujitsu, Ltd.
- GoAhead Software, Inc.
- HADCO Corporation
- HCL Infosystems Ltd.
- Hewlett-Packard Company
- Hewlett-Packard GmbH

- Hitachi Ltd.
- Hybricon Corporation
- I-Bus/Phoenix Corporation
- InnoMediaLogic, Inc.
- Intel Corporation
- Integra Micro Systems (P) Ltd.
- Interphase Corporation
- InterWorks Computer Products
- Inventec Corporation
- Ipex ITG
- JMC Products
- Kaparel Corporation
- L-3 Communications Corp.
- Legend (Beijing) Limited
- Linux NetworX, Inc.
- Lynux Works, Inc.
- Macrolink, Inc
- Magnetek, Inc.
- Micro-Star International
- Mirapoint, Inc.
- MiTAC International Corp.
- Mitsubishi Electric Corp.
- Information Systems Engineering Center
- Motorola Computer Group
- National Semiconductor Corp.
- NEC Corporation
- Nematron Corporation
- Network Appliance, Inc.
- Network Engines, Inc.
- Network Storage Solutions, Inc.
- NOCpulse, Inc.
- Olivetti Computers Worldwide
- OSA Technologies
- PEP Modular Computers
- Performance Technologies, Inc.
- Phoenix Technologies Ltd.
- Pigeon Point Systems
- Pinnacle Data Systems, Inc.
- Prais, Inc.
- Qlogic Corporation
- Quanta Computer Inc.

- Radisys Corporation
- RAMIX Inc.
- Reliance Computer Corporation
- Samsung Electronics Co., LTD
- Sanera Systems, Inc.
- SANGate Systems, Inc.
- SBS Technologies, Inc.
- Scenix Semiconductor, Inc.
- Siemens AG
- Silicon Graphics, Inc.
- SKY Computers, Inc.
- Stan Cox & Associates
- Standard Microsystems Corporation
- StrataLight Communications, Inc.
- Stratus Computer Systems Ireland Ltd.
- Summit Microelectronics, Inc.
- Sun Microsystems
- Super Micro Computer, Inc.
- Symphony Group Intl. Co., Ltd.
- Synergy Microsystems
- Teknor Applicom, Inc.
- T-Netix, Inc.
- Tatung Co.
- Tektronix
- Texas Micro Corporation
- Toshiba Corporation
- Trimm Technologies
- Trilogic Systems, LLC
- Tyan Computer Corporation
- Universal Scientific Industrial Corp.
- USAR Systems, Inc.
- VIA Technologies, Inc.
- Vitesse Semiconductor Corp.
- Vividon, Inc.
- Vooha, Inc.
- Watrin System Design
- Winbond Electronics Corp.
- Wistron Corporation
- Ziatech Corporation
- ZNYX Networks, Inc.

Server Management Controllers, Sensors, and Tools

Tom Slaight
Intel Corporation



Questions?

Please remember to turn in
your session survey form.

**This presentation will be
posted September 26th**

<http://www.intel.com/idf>

**Attendee password will be sent two weeks
after the conference via email.**

Vitesse Semiconductor* VSC055

I²C Enhanced Backplane Controller

- Designed as a companion component to enclosure management processors
 - I²C bus
 - 8 Fan-speed monitor inputs
 - 8 Programmable PWM outputs
 - 64 12mA, programmable, bi-directional I/O pins with individually selectable one of 7 LED flash rates
 - 32 GPIO pins can be used as FC-AL port bypass control pins
 - Programmable interrupt control for 64 interrupt sources (I/O, bypass input transitions, and fan speed thresholds)
 - 100-pin PQFP pkg.
- SSC050 is a subset of the VSC055 (e.g. 4 fans & PWM, 40 GPIO, 52 interrupt sources)*

Intel **Developer** Forum Fall 2002

https://web.archive.org/web/20031002054319/http://www.intel.com/design/servers/ipmi/RMCP+_and_SOL_Proposals.pdf

- IPMI -

Intelligent Platform Management Interface Specification

Proposed extensions for IPMI v2.0:

RMCP+ and Serial Over LAN

DRAFT 9/15/03

Intel Hewlett-Packard NEC Dell

Revision History

Date	Modifications
9/15/03	Release for Adopter review

Copyright © 2003 Intel Corporation, Hewlett-Packard Company, NEC Corporation, Dell Computer Corporation, All rights reserved.

INTELLECTUAL PROPERTY DISCLAIMER

THIS SPECIFICATION IS PROVIDED “AS IS” WITH NO WARRANTIES WHATSOEVER INCLUDING ANY WARRANTY OF MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE, OR ANY WARRANTY OTHERWISE ARISING OUT OF ANY PROPOSAL, SPECIFICATION, OR SAMPLE.

NO LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE, TO ANY INTELLECTUAL PROPERTY RIGHTS IS GRANTED OR INTENDED HEREBY.

INTEL, HEWLETT-PACKARD, NEC, AND DELL DISCLAIM ALL LIABILITY, INCLUDING LIABILITY FOR INFRINGEMENT OF PROPRIETARY RIGHTS, RELATING TO IMPLEMENTATION OF INFORMATION IN THIS SPECIFICATION. INTEL, HEWLETT-PACKARD, NEC, AND DELL, DO NOT WARRANT OR REPRESENT THAT SUCH IMPLEMENTATION(S) WILL NOT INFRINGE SUCH RIGHTS.

I²C is a trademark of Philips Semiconductors. All other product names are trademarks, registered trademarks, or servicemarks of their respective owners.

I²C is a two-wire communications bus/protocol developed by Philips. IPMB is a subset of the I²C bus/protocol and was developed by Intel. Implementations of the I²C bus/protocol or the IPMB bus/protocol may require licenses from various entities, including Philips Electronics N.V. and North American Philips Corporation.

Intel, Hewlett-Packard, NEC, and Dell retain the right to make changes to this document at any time, without notice. Intel, Hewlett-Packard, NEC, and Dell make no warranty for the use of this document and assume no responsibility for any error which may appear in the document nor does it make a commitment to update the information contained herein.

Table of Contents

1. Introduction	6
1.1 Reference Documents	6
1.2 SOL Overview	7
1.3 RMCP+ Overview	7
2. RMCP+ Protocols	7
2.1 Differences with IPMI v1.5 Authentication	8
2.2 IPMI v1.5 Discovery and Sessions under IPMI v2.0/RMCP+	9
2.3 Support for IPMI v2.0 Capabilities under IPMI v1.5	9
2.4 VLAN Support	9
2.5 IPMI Messages Encapsulation Under RMCP	9
2.5.1 RMCP/ASF and IPMI Byte Order	10
2.6 Example IPMI over LAN Packet	11
2.7 Discovering Support For IPMI over IP Connections	14
2.8 Channel Security Discovery Commands	14
2.9 Get Channel Authentication Capabilities Command	14
2.10 Get Channel Cipher Suites Command	17
2.10.1 Cipher Suite Records	18
2.10.2 Cipher Suite IDs	19
2.11 RMCP/ASF Pong Message (Ping Response)	20
2.12 AES-CBC-128 Encrypted Payloads	21
2.12.1 Generating the Initialization Vector	21
2.12.2 Encryption with AES	22
2.12.3 CBC (Cipher Block Chaining)	22
2.13 RMCP+ Authentication	22
2.13.1 RSSP Open Session Request	22
2.13.2 RSSP Open Session Response	24
2.14 Close Session Command	25
2.15 Session Termination	25
2.16 RAKP Messages	26
2.16.1 RAKP Message 1	26
2.16.2 RAKP Message 2	27
2.16.3 RAKP Message 3	28
2.16.4 RAKP Message 4	29
2.17 RSSP and RAKP Message Status Codes	30
2.18 IPMI v2.0 RMCP+ Payload Types	31
2.19 Session Sequence Numbers	32
2.20 IPMI v1.5 Session Sequence Number Handling	32
2.20.1 IPMI v1.5 Inbound Session Sequence Number Tracking and Handling	33
2.20.2 IPMI v1.5 Out-of-order Packet Handling	33
2.20.3 IPMI v1.5 Outbound Session Sequence Number Tracking and Handling	34
2.21 IPMI v2.0 RMCP+ Session Sequence Number Handling	34
2.21.1 IPMI v2.0 RMCP+ Sliding Window	35
2.22 Authentication, Integrity, and Encryption Algorithm Numbers	35
2.23 RSSP Authenticated Key-Exchange Protocol (RAKP)	36
2.24 Generating Additional Keying Material	39
2.25 Random Number Generation	40
2.25.1 Random Number Key	40
2.25.2 Random Number Generator Counters	40
2.25.3 Random Number Generator Operation	40
2.26 Extended LAN Configuration Data	41
3. RMCP+ Support and Payload Commands	43
3.1 Activate Payload Command	43

3.2	Deactivate Payload Command	45
3.3	Suspend/Resume Payload Encryption Command	45
3.4	Get Payload Activation Status Command	46
3.5	Get Payload Instance Info Command	47
3.6	Set User Payload Access Command	48
3.7	Get User Payload Access Command	50
3.8	Get Channel Payload Support Command	51
3.9	Get Channel Payload Version Command	51
3.10	Get Channel OEM Payload Info Command	52
3.11	Set User Password Command	52
3.12	Set Channel Security Keys Command	54
4.	Serial Over LAN.....	55
4.1	SOL and Serial Port Sharing	56
4.2	SOL Operation Overview.....	57
4.3	SOL Security.....	58
4.4	SOL Sequence Numbers.....	58
4.5	Flow Control.....	58
4.6	Bit Rate Handling.....	58
4.7	SOL Payload Data Format	59
4.8	Activating SOL using RMCP+ Authentication	61
4.9	SOL Packet Acknowledge and Retries	62
4.10	SOL Interaction with Windows.NET Escape Sequences	64
4.11	SOL Payload Activated with Serial Port Sharing.....	64
5.	SOL Commands	66
5.1	SOL Activating Command	66
5.2	Set SOL Configuration Parameters Command	67
5.3	Get SOL Configuration Parameters Command.....	67
	Appendix A - Previous Sequence Number Tracking	71
	Appendix B - Command Assignments	72

List of Tables

Table 2-1, RMCP Packet Format for IPMI via Ethernet.....	11
Table 2-2, Channel Security Discovery Commands	14
Table 2-3, Get Channel Authentication Capabilities Command	15
Table 2-4, Get Channel Cipher Suites Command	18
Table 1-4, Cipher Suite Record Format	19
Table 2-6, Cipher Suite IDs.....	19
Table 2-7, RMCP Packet Fields for ASF Presence Pong Message (Ping Response).....	20
Table 2-8, AES-CBC Encrypted Payload Fields.....	21
Table 2-9, RSSP Open Session Request.....	23
Table 2-10, RSSP Open Session Response	24
Table 2-11, Close Session Command.....	25
Table 2-12, RAKP Message 1	26
Table 2-13, RAKP Message 2	28
Table 2-14, RAKP Message 3	29
Table 1-14, RAKP Message 4.....	30
Table 1-15, RSSP and RAKP Message Status Codes	30
Table 2-17, Payload Type Numbers.....	32
Table 2-18, Authentication Algorithm Numbers.....	35
Table 2-19, Integrity Algorithm Numbers.....	35

Table 2-20, Confidentiality Algorithm Numbers	36
Table 2-21, Extended LAN Configuration Data	41
Table 3-1, RMCP+ Support and Payload Commands	43
Table 3-2, Activate Payload Command.....	44
Table 3-3, Deactivate Payload Command	45
Table 3-4, Payload-specific Encryption Behavior.....	46
Table 3-5, Suspend/Resume Payload Encryption Command	46
Table 3-6, Get Payload Activation Status Command.....	47
Table 3-7, Get Payload Instance Info Command	48
Table 3-8, Set User Payload Access Command	49
Table 3-9, Get User Payload Access Command.....	50
Table 3-10, Get Channel Payload Support Command	51
Table 3-11, Get Channel Payload Version Command	52
Table 3-12, Get Channel OEM Payload Info Command.....	52
Table 3-13, Set User Password Command	54
Table 3-14, Set Channel Security Keys Command	55
Table 4-1, Mux Settings	56
Table 4-2, SOL Payload Data Format	59
Table 4-3, Remote Console to BMC SOL Packet Handling	63
Table 4-4, Set Serial/Modem Mux Command Operation while SOL Active.....	65
Table 5-1, SOL Commands.....	66
Table 5-2, SOL Activating Command.....	66
Table 5-3, Set SOL Configuration Parameters Command	67
Table 5-4, Get SOL Configuration Parameters Command.....	67
Table 5-5, SOL Configuration Parameters	68

1. Introduction

This document presents proposed additions to the IPMI specification. At this time, it is planned that these proposals will be included in revised IPMI specification that will be issued as “IPMI v2.0”. The document is being provided to IPMI Adopters for comment.

The proposals in this document are provided for information only. Within this document the term “IPMI v2.0” is used to denote or highlight elements that are different or in addition to the present IPMI v1.5 specification. It does not imply that the proposals have been formally reviewed and accepted by the IPMI Promoters, nor whether they will be included in any future IPMI specification in part or in whole. All proposals are subject to review and approval by the IPMI Promoters (Intel, HP, NEC and Dell) and are subject to change without prior notice.

The proposals in this document primarily cover the following areas: RMCP+ and SOL. Other proposal areas that have been publicly discussed are being worked on as well. These areas were not available in suitable form for IPMI Adopter review at this time.

1.1 Reference Documents

The following documents are companion and supporting specifications for this document:

- [AES] *Advanced Encryption Standard*, FIPS 197, November 2001.
<http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>

- [ASF 2.0] *Alert Standard Format (A S F) Specification Version 2.0*, 23 April 2003,
©2000-2003, Distributed Management Task Force, Inc.
<http://www.dmtf.org>

- [RFC2104] *RFC 2104, HMAC: Keyed-Hashing for Message Authentication*, H.
Krawczyk, IBM, M. Bellare, UCSD, R. Canetti, IBM, February 1997.
<http://www.ietf.org/rfc/rfc2104.txt>

- [RFC2404] *RFC 2404, The Use of HMAC-SHA-1-96 within ESP and AH*, C. Madson,
Cisco Systems Inc., R. Glenn, NIST, November 1998.
<http://www.ietf.org/rfc/rfc2404.txt>

- [SHA-1] NIST, FIPS PUB 180-2: Secure Hash Standard, August 2002.
<http://csrc.nist.gov/publications/fips/fips180-2/fips180-2.pdf>

- [MODES] *Recommendation for Block Cipher Modes of Operation: Methods and
Techniques*, NIST Special Publication 800-38A, December 2001.
<http://csrc.nist.gov/publications/nistpubs/800-38a/sp800-38a.pdf>

[MSFT EMS] *Building Hardware and Firmware to Complement Microsoft Windows Headless Operation, Version 1.00, July 16, 2002.*
<http://www.microsoft.com/whdc/hwdev/platform/server/headless/>

1.2 SOL Overview

SOL stands for “Serial over LAN” and is the working name for the specification of IPMI interfaces for redirecting baseboard serial controller traffic over an IPMI session via RMCP+. This can be used to enable asynchronous serial-based OS and pre-OS communication over a connection to the BMC. SOL is specified in *Section 4, Serial Over LAN* and *Section 4.10, SOL Interaction with Windows.NET Escape Sequences*.

1.3 RMCP+ Overview

RMCP+ is the working name for an enhanced protocol for transferring IPMI messages and other types of payloads to an IPMI-based BMC over IP. RMCP+ is specified in *Section 2, RMCP+ Protocols*, and *Section 3, RMCP+ Support and Payload Commands*.

RMCP+ includes:

- **Support for multiple payload types over an IPMI session.** These include both ‘standard’ and ‘OEM’ payloads. Payload support enables types of traffic other than IPMI messages to be simultaneously carried over an IPMI session. The specification also allows a separate session to be established for carrying payloads.
- **Enhanced user authentication algorithms.** RMCP+ includes more robust session set-up and key handling algorithms than those for IPMI over LAN in IPMI v1.5.
- **Encryption support.** IPMI messages and other payloads can be encrypted under an IPMI session. This enables confidentiality for remote operations such as setting user passwords and for SOL.
- **Better alignment with the ASF 2.0.** The RMCP+ follows many of the packet format and authentication elements defined for RMCP (Remote Management Control Protocol) as specified in the Distributed Management Task Force “ASF 2.0” specification. (See [ASF 2.0] in the *Reference Documents* section.) This helps make it easier to create remote management applications that can work with both ASF 2.0 and IPMI v2.0 -based remote management connections.

2. RMCP+ Protocols

This section presents the packet formats, algorithms, and protocols associated with RMCP+. The following lists the main characteristics of RMCP+:

- Uses an ASF 2.0 -based authentication algorithms and steps for establishing a session. (Commands corresponding to the ASF 2.0 RSP/RAKP messages, however, are defined under the IPMI message class rather than under the ASF message class. The commands vary slightly from their ASF 2.0 counterparts, but are close enough to

make it straightforward to create a single set of code that can connect to either IPMI or ASF -based systems.)

- Supports encrypted payloads.
- Authenticated sessions can be established on any UDP port, including port 26Fh
- Multiple payload types are supported over a single session
- A mix of authenticated and unauthenticated packets can be supported over a single session
- RMCP+ is specified under the IPMI message class in RMCP. The packets remain conformant with the RMCP specification at the RMCP packet level.
- There can be a mix of encrypted and unencrypted payloads over a single session

2.1 Differences with IPMI v1.5 Authentication

The following presents an overview of some of the key differences as well as similarities between IPMI v1.5 and RMCP+ user setup and authentication mechanisms:

- IPMI v1.5 had a hook that would allow packets under a session to have different authentication signatures than the type of signature that was negotiated to open the session. This hook was to include a 'Authentication Type' field that specified the type of authentication on a per-packet basis. This capability was not used in the specification. Thus, to simplify things a packet can only have two types of authentication: the type of authentication selected in the "Open Session" command or "none" - thus the 'Authentication Type' field is deleted and instead the presence or absence of the "Integrity Data" field is used to indicate whether a given packet in the session is authenticated or not.
- For RMCP+, the IPMI *Get Session Challenge* request and response messages and *Activate Session* messages are replaced by the *Open Session* and *RAKP_x* Messages:
 1. The RMCP *RSSP Open Session Request*, *RSSP Open Session Response*, and *RAKP_1* messages, combined, correspond roughly to the IPMI *Get Session Challenge* request.
 2. The RMCP *RAKP_2* message corresponds roughly to an IPMI *Get Session Challenge* response.
 3. The RMCP *RAKP_3* message corresponds roughly to an IPMI *Activate Session* request.
 4. Note that there's no corresponding response message to acknowledge that the *RAKP_3* message. However, as an ASF message there will be an RMCP ACK that is sent back if the RMCP sequence number is not FFh.
- ASF 2.0 authentication defined 'Roles' such as User and Administrator, where a key (password) was associated with each role. IPMI v1.5 authentication associated a key with each User Name where a 'privilege level' (such as User or

Administrator) was configured for each user name. These two approaches are merged in RMCP+. The BMC can be configured with 'null' user names, whereby key lookup is done based on 'privilege level only', or with non-null user names, where the key lookup for the session is determined according to the user name.

- RMCP+ supports HMAC using SHA-1 for authentication. We expect that this type of authentication will be significantly slower than IPMI v1.5 Authentication for a given processor (which uses a simple MD2 or MD5 signature). First, RMCP uses SHA-1 which, while more robust, is computationally more intensive. Probably more significant is the use of HMAC which requires that the hashing algorithm is run twice on the data for each packet.

IPMI v1.5	Single Session ID for BMC and console	BMC and remote console pick unique, random starting 'incoming' and 'outgoing' sequence numbers
RMCP+	Separate Session IDs for BMC and console	BMC and remote console pick their own session IDs.

2.2 IPMI v1.5 Discovery and Sessions under IPMI v2.0/RMCP+

To enable backward compatibility, an IPMI v2.0 BMC that supports RMCP+ for IPMI messaging and standard payloads is required to simultaneously support IPMI v1.5 Packet formats. For a given BMC, users must be equally configurable for IPMI v2.0 or v1.5 access.

2.3 Support for IPMI v2.0 Capabilities under IPMI v1.5

IPMI v2.0-specific capabilities related to payloads (e.g. Serial Over LAN) are only available over IPMI v2.0/RMCP+ sessions. Otherwise, new IPMI v2.0 commands and command extensions are available under IPMI v1.5 sessions.

2.4 VLAN Support

VLAN support is optional, but recommended, for BMC access via IPMI 1.5 packet and IPMI v2.0 packet formats. A BMC that supports VLAN on a channel is required to support it for both packet formats. A BMC is not required to support VLAN equally across all LAN channels. This is allowed because some LAN connections may not have hardware that supports VLAN with a LAN connection to the BMC.

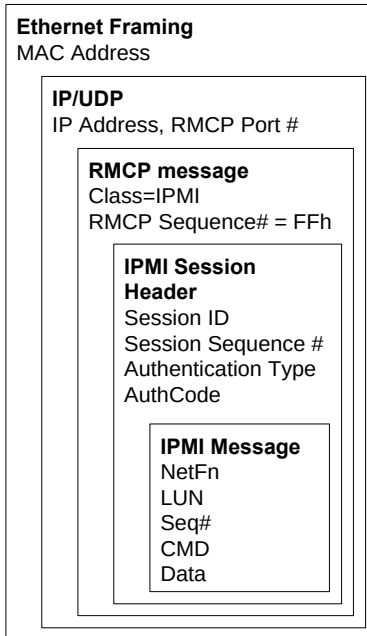
When a VLAN ID is configured into the LAN parameters, the BMC will only accept packets with that VLAN tag. This includes all RMCP and RMCP+ packets as well as DHCP and ARP packets. Conversely, all BMC-generated packets will include the given VLAN tag.

2.5 IPMI Messages Encapsulation Under RMCP

For LAN transfers, IPMI messages are a special class of data encapsulated in an IPMI Session packet. The IPMI Session packets are encapsulated in RMCP packets, which are encapsulated in UDP datagrams. This is illustrated in the following figure. The same type of encapsulation is used for IPMI

serial/modem messages via PPP, except the Ethernet Framing is replaced with a packet that uses PPP Framing and IP protocol type.

Figure 2-1, IPMI LAN Packet Layering



2.5.1 RMCP/ASF and IPMI Byte Order

Please take note of the following:

Multi-byte fields in RMCP/ASF fields are specified as being transmitted in 'Network Byte Order' - meaning most-significant byte first.

RMCP and ASF-specified fields are therefore transferred most-significant byte first.

The IPMI convention is to transfer multi-byte numeric fields least-significant Byte first. Therefore, unless otherwise specified:

Data in the IPMI Session Header and IPMI Message fields are transmitted least-significant byte first.

2.6 Example IPMI over LAN Packet

The following table shows the format and fields for IPMI messages encapsulated in an RMCP packet that is itself encapsulated within an IPv4 UDP packet delivered over Ethernet.

The table includes a set of columns under the label “Format”. The first column shows the format used for IPMI over LAN as originally defined in the IPMI v1.5 specification, the second column shows the format for authenticated packets delivered using RMCP per the ASF 2.0 specification. This column is shown for reference only. The third column shows the format of packets for RMCP+.

Table 2-1, RMCP Packet Format for IPMI via Ethernet

Field	Format			Value
	RMCP / IPMI 1.5 26Fh	ASF RMCP 298h	RMCP / IPMI 2.0 “RMCP+” 26Fh	
Destination Address	6			Dest. MAC Address for 802.3
Source Address	6			Source MAC Address for 802.3
802.1q ^[10] TPI	2			Tag Protocol Identifier =8100h
VLAN TAG - user priority	3-bits			User Priority ^[11] default = 000b, configurable
VLAN TAG - CFI	1-bit			Canonical Format Indicator = 0b
VLAN TAG - VLAN ID	12-bits			0's = no VLAN
Frame Type	2			0800h
IP Header	Version	4-bits		4h for IPv4
	Header Length (length of IP header in units of 4- bytes)	4-bits		5h
	Precedence	3-bits		000b ^[4]
	Service Type (Type of Service)	4-bits		1000b ^[4] (minimize delay)
	reserved	1-bit		0b
	Total Length	2		
	Identification ^[5]	2		note ^[5]
	Flags	3-bits		010b ^[6] (don't fragment)
	Fragment Offset	13-bits		0_0000_0000_0000b ^[7]
	Time-to-Live	1		40h ^[3]
	Protocol	1		11h
	Header Checksum	2		
	Source IP Address	4		
	Destination IP Address	4		(28 bytes)
UDP Header	Source Port	2		
	Destination Port	2		26Fh, 298h
	UDP Length	2		
	UDP Checksum	2		(36 bytes) Some payloads may not use the UDP checksum, in which case this field gets set to 0's. The receiver must accept the packet and ignore the checksum field in this case.
RSP Header	Session ID		4	
	Session Sequence #		4	

RMCP Header	Version	1	1	1	06h (ASF 2.0)
	Reserved	1	1	1	
	RMCP Seq #	1	1	1	FFh for IPMI ^[2]
	Class of Message	1	1	1	07h for IPMI
IPMI Session Header	Auth Type / Format	1		1	[7:4] - reserved [3:0] - Authentication Type / Format 0h = none 1h = MD2 2h = MD5 3h = reserved 4h = straight password / key 5h = OEM proprietary 6h = Format = RMCP+ (IPMI v2.0 only) all other = reserved
	Payload Type			1	Payload Type [7] - 0b = payload is unencrypted 1b = payload is encrypted [6] - 0b = payload is unauthenticated (no AuthCode field) 1b = payload is authenticated (AuthCode field is present) [5:0] = payload type. See Table 2-17, Payload Type Numbers.
	OEM IANA			4	This field is only present when Payload Type = 02h (OEM Explicit) byte 1:3 - OEM IANA byte 4 - reserved
	OEM Payload ID			2	This field is only present when Payload Type = 02h (OEM Explicit). The definition and values of this field are specified by the company or body identified by the OEM IANA field.
	IPMI v2.0 RMCP+ Session ID			4	note ^[8] Session ID is 0000_0000h for messages that are sent 'outside' of a session.
	Session Sequence Number	4		4	note ^[8] For IPMI v2.0 "RMCP+" there are separate sequence numbers tracked for authenticated and unauthenticated packets. 0000_0000h is used for packets that are sent 'outside' of a session.
	IPMI v1.5 Session ID	4			note ^[8] Session ID is 0000_0000h for messages that are sent 'outside' of a session.
	Msg. Auth. Code Code (AuthCode) (not present when Authentication Type set to 'none'.)	16			
	IPMI Msg/Payload length	1	2	2	Payload length in bytes. 1-based.
	Confidentiality Header			var	For encrypted payloads, based on encryption type for given payload. The confidentiality header is not encrypted.
IPMI Payload	Payload Data	var	var	var	For IPMI v2.0: IPMI, SOL, KVM,

IPMI Session Trailer ^[9] / RSP Trailer				etc. per Payload Type field.
	Confidentiality Trailer		var	For encrypted payloads, based on encryption type for given payload. The confidentiality trailer is typically encrypted along with the Payload Data.
	Integrity PAD		var	Added as needed to cause the number of bytes in the data range covered by the AuthCode (Integrity Data) field to be a multiple of 4 bytes (DWORD). If present, each Integrity Pad byte is set to 00h.
	Pad Length		1	indicates how many pad bytes were added so that the amount of non-pad data can be determined.
	Next Header		1	Reserved in IPMI v2.0. Set to 00h when writing, ignore when read.
	AuthCode (Integrity Data)		var	For IPMI v1.5 this field is as specified by Auth Type. For IPMI v2.0 (RMCP+) if this field is present, then it is calculated according to the Integrity Algorithm that was negotiated during the session open process. See Table 2-19, <i>Integrity Algorithm Numbers</i> . This field is absent when the packet is unauthenticated.
MAC level	Legacy PAD ^[1]	1		legacy PAD not needed for IPMI v2.0
	CRC	4	4	4

1. Some LAN adapter chips may have a problem where packets of overall lengths 56, 84, 112, 128, or 156 are not handled correctly. The PAD byte is added as necessary to avoid these overall lengths. Remote console software must use the PAD byte when formatting packets to any 10/100 Ethernet device that accepts RMCP packets.
2. RMCP Messages with class=IPMI should be sent with an RMCP Sequence Number of FFh to indicate that an RMCP ACK message should not be generated by the message receiver.
3. Default value for packets transmitted from the BMC. Can be overridden via a configuration parameter setting.
4. Value used for packets transmitted from the BMC. The BMC ignores the value of this parameter (except for checksum calculations) on received packets.
5. BMC should increment this field each time it sends a new packet.
6. Default value for packets transmitted from the BMC. Bit offset 1 (fragment bit) can be overridden via a configuration parameter setting.
7. Default value for packets transmitted from the BMC. The BMC is not required to support receiving fragmented packets. Packets with a non-zero fragment offset and/or a flags field bit 2 = 1b ("more fragments" may be silently discarded.)
8. The Session ID and Session Sequence Number must be non-zero for commands executed during an active session. All 0's (0000_0000h) for the Session ID and/or Session Sequence Number (null Session ID, null Session Sequence Number) are special values only used for messages and commands that can be executed prior to establishing a session, e.g. Get System GUID, Get Channel Authentication Capabilities, Get Session Challenge, and RSSP, or RAKP messages. When the Session ID is 0000_0000h, the Sequence Number field is ignored, however the Session Sequence Number should still be set to 0000_0000h. In addition, for IPMI v2.0 RMCP+ packets, unless otherwise indicated bits 7:6 in the payload type field must also indicate that the packet is both unauthenticated and unencrypted. Note that the IPMI v1.5 Activate Session uses a null (all 0's) Session Sequence Number before a session is activated, but does not use a null Session ID. Instead, it uses the Temporary Session ID given by the BMC in the response to the Get Session Challenge command.
9. For IPMI v2.0 RMCP+ packets, the IPMI Session Trailer is absent whenever the Session ID is 0000_0000h, or whenever bit 6 in the payload type field indicates the packet is unauthenticated
10. Four bytes only present for IEEE 802.1q "VLAN" formatted packets

11. The use and interpretation of this number is defined in ISO/IEC 15802-3.

2.7 Discovering Support For IPMI over IP Connections

There are two mechanisms for discovering whether a given system (IP Address) supports IPMI v1.5 and/or IPMI v2.0 connections. The first is the ‘RMCP Ping discovery’ mechanism where the BMC returns whether IPMI is supported via the *supported entities* field of the Ping response (a.k.a. the ‘Pong’ message). The BMC can optionally return values in the Ping response indicating what level of IPMI connection(s) (IPMI v1.5 and/or IPMI v2.0) it supports.

The second mechanism is an ‘IPMI command discovery’ mechanism where the remote console discovers that the system supports IPMI by issuing a *Get Channel Authentication Capabilities* command to determine support for IPMI v1.5/IPMI v2.0 connections. BMCs that support IPMI v2.0/RMCP+ must support the *Get Channel Authentication Capabilities* command in both the IPMI v1.5 and IPMI v2.0 packet formats. It is recommended that a remote console use the IPMI v1.5 formats until it has confirmed IPMI v2.0 support.

When the remote console decides to connect to the discovered system, it can use the *Get Channel Authentication Capabilities* and (for IPMI v2.0) the *Get Channel Cipher Suites* commands to determine which authentication, integrity, and confidentiality algorithm Cipher Suite(s) can be used for establishing the connection.

2.8 Channel Security Discovery Commands

Table 2-2, Channel Security Discovery Commands

Command	Section Defined	O/M
Get Channel Authentication Capabilities	2.9	O ^[1]
Get Channel Cipher Suites	2.10	O ^[2]

1. Mandatory if session-based channels are implemented.

2. Mandatory if IPMI v2.0 / RMCP+ session-based channels are implemented.

2.9 Get Channel Authentication Capabilities Command

This command is sent in unauthenticated (clear) format. This command is used to retrieve capability information about the channel that the message is delivered over, or for a particular channel. The command returns the authentication algorithm support for the given privilege level. When activating a session, the privilege level passed in this command will normally be the same Requested Maximum Privilege level that will be used for a subsequent *Activate Session* command.

BMC implementations of IP-based channels must support the *Get Channel Authentication Capabilities Command* using the IPMI v1.5 packet format. BMCs that support IPMI v2.0 RMCP+ must also support the command using the IPMI v2.0/RMCP+ format.

The *Get Channel Authentication Capabilities* command can also be used as a no-op 'Ping' to keep a session from timing out.

Table 2-3, Get Channel Authentication Capabilities Command

Session Request Data		authentication type = NONE
		session seq# = null (0's)
		session ID = null (0's)
		AuthCode = NOT PRESENT
IPMI Request Data	1	Channel Number [7] - 1b = get IPMI v2.0+ extended data. 0b = Backward compatible with IPMI v1.5. Result data only returns bytes 1:9, bit [7] of byte 3 (Authentication Type Support) returns as 0b, byte 5 returns 00h. [6:4] - reserved [3:0] - channel number. 0h-7h, Fh = channel numbers Eh = retrieve information for channel this request was issued on.
	2	Requested Maximum Privilege Level [7:4] - reserved [3:0] - requested privilege level 0h = reserved 1h = Callback level 2h = User level 3h = Operator level 4h = Administrator level 5h = OEM Proprietary level

Session Response Data		authentication type = NONE
		session seq# = null (0's)
		session ID = null (0's)
		AuthCode = NOT PRESENT
IPMI Response Data	1	Completion Code
	2	Channel Number Channel number that the Authentication Capabilities is being returned for. If the channel number in the request was set to Eh, this will return the channel number for the channel that the request was received on.
	3	Authentication Type Support Returns the setting of the Authentication Type Enable field from the configuration parameters for the given channel that corresponds to the Requested Maximum Privilege Level. [7] - 1b = IPMI v2.0+ extended capabilities available. See Extended Capabilities field, below. 0b = IPMI v1.5 support only. [6] - reserved [5:0] - IPMI v1.5 Authentication type(s) enabled for given Requested Maximum Privilege Level All bits: 1b = supported 0b = authentication type not available for use. [5] - OEM proprietary (per OEM identified by the IANA OEM ID in the RMCP Ping Response) [4] - straight password / key [3] - reserved [2] - MD5 [1] - MD2 [0] - none
	4	[7:5] - reserved Following bits apply to IPMI v1.5 and v2.0: [4] - Per-message Authentication status 0b = Per-message Authentication is enabled. Packets to the BMC must be authenticated per Authentication Type used to activate the session, and User Level Authentication setting, following. 1b = Per-message Authentication is disabled. Authentication Type 'none' accepted for packets to the BMC after the session has been activated. [3] - User Level Authentication status 0b = User Level Authentication is enabled. User Level commands must be authenticated per Authentication Type used to activate the session. 1b = User Level Authentication is disabled. Authentication Type 'none' accepted for User Level commands to the BMC. [2:0] - Anonymous Login status This parameter returns values that tells the remote console whether there are users on the system that have 'null' usernames. This can be used to guide the way the remote console presents login options to the user. (see IPMI v1.5 specification sections 6.9.1, <i>'Anonymous Login' Convention</i> and 6.9.2, <i>Anonymous Login</i>) [2] - 1b = Non-null usernames enabled. (One or more users are enabled that have non-null usernames). [1] - 1b = Null usernames enabled (One or more users that have a null username, but non-null password, are presently enabled) [0] - 1b = Anonymous Login enabled (A user that has a null username and null password is presently enabled)

5	For IPMI v1.5: - reserved For IPMI v2.0+: - Extended Capabilities [7:2] - reserved [1] - 1b = channel supports IPMI v2.0 connections. [0] - 1b = channel supports IPMI v1.5 connections.
6:8	OEM ID IANA Enterprise Number for OEM/Organization that specified the particular OEM Authentication Type. Least significant byte first. Return 00h, 00h, 00h if no OEM authentication type available.
9	OEM auxiliary data. Additional OEM-specific information for the OEM Authentication Type. Return 00h if no OEM authentication type available.

2.10 Get Channel Cipher Suites Command

This command can be executed prior to establishing a session with the BMC. The command is used to look up what authentication, integrity, and authentication algorithms are supported. The algorithms are used in combination as ‘Cipher Suites’.

The data is accessed 16-bytes at a time starting from List Index field value of 0 in the request and then repeating the request incrementing the List Index field each time until fewer than 16-bytes of algorithm data (or no algorithm data) is returned in the response, or the maximum List Index value has been reached.

A given Cipher Suite may only be available for establishing a session at a particular maximum privilege level or lower. For example, a Cipher Suite that has a privilege level of ‘Admin’ can therefore be used for any privilege level, while a privilege level of User can only be used for establish sessions with a Maximum Requested Privilege Level of User or Callback.

Because the authentication algorithm specifies the steps for authenticating the user, it is a necessary part of session establishment. Therefore an authentication algorithm number is required for all Cipher Suites. It is possible that a given Cipher Suite may not specify use of an integrity or confidentiality algorithm. If the Cipher Suite has integrity and/or confidentiality of ‘none’, then all the same steps for establishing a session are used (open session request/response, RAKP messages) - but the integrity (AuthCode) and confidentiality fields will be absent in packets for that are sent under the session.

Table 2-4, *Get Channel Cipher Suites Command*

IPMI Request Data	1	Channel Number [7:4] - reserved [3:0] - channel number. 0h-7h, Fh = channel numbers Eh = retrieve information for channel this request was issued on.
	2	Payload Type. [7:6] - reserved [5:0] - Payload Type number Typically 00h (IPMI). The Payload Type number is used to look up the Security Algorithm support when establishing a separate session for a given payload type.
	3	List Index. [7] - 1b = list algorithms by Cipher Suite 0b = list supported algorithms [6] - reserved [5:0] - List index (00h-3Fh) 00h = get first set of algorithm numbers. BMC returns up to sixteen (16) bytes per index. 0h selects the first set of 16, 1h selects the next set of 16, and so on. The BMC will return 0 bytes of supported algorithm numbers when the indexed range exceeds the number of available algorithms. A return of 0 bytes of supported algorithm numbers, or a return of fewer than 16 numbers indicates that the full list has been read. Note that Cipher
IPMI Response Data	1	Completion Code
	2	Channel Number Channel number that the Authentication Algorithms are being returned for. If the channel number in the request was set to Eh, this will return the channel number for the channel that the request was received on.
	(3:18)	Cipher Suite Record data bytes, per <i>Table 1-4, Cipher Suite Record Format</i> . Record data is 'packed'; there are no pad bytes between records. It is possible that record data will span across multiple List Index values. The BMC returns Cipher Suite Record data 16-bytes at a time, starting from List Index 00h.

2.10.1 Cipher Suite Records

The data from the *Get Channel Cipher Suites* command is issued as Cipher Suite records. Tag bits are used to delimit different fields in the record. Each record starts off with header bytes that are tagged with the value 11b (for this version of the specification, there is only one header byte). Following the header bytes are algorithm number bytes for the different algorithms that form the Cipher Suite. Each byte is tagged with the type of algorithm the number is for. Cipher Suite records are required to list algorithms in the order: Authentication Algorithm number first, Integrity Algorithm numbers next, and Confidentiality Algorithm numbers last.

If more than one algorithm of a given type is listed in the Cipher Suite Record, then any one of the algorithms can be used in combination with the other types. For example, if a Cipher Suite response returns both MD5 and MD2 as Authentication and Integrity algorithms, and !RC4 for confidentiality, then the allowed combinations are [MD2, MD2, !RC4], [MD2, MD5, !RC4], [MD5, MD2, !RC4], and [MD5, MD5, !RC4]. I.e. a remote console can negotiate for those combinations when establishing a session.

Table 1-4, Cipher Suite Record Format

bytes	Tag bits [7:6]	Data bits [5:0]
1	11b	Header. <u>Byte 1 - Cipher Suite ID.</u> This value is used a numeric way of identifying the Cipher Suite on the platform. It's used in commands and configuration parameters that enable and disable Cipher Suites. See <i>Table 2-6, Cipher Suite IDs</i> .
(2:4)	11b	<u>Byte 1:3 - OEM IANA</u> Least significant byte first. These bytes are present in the record for an OEM Cipher Suite number. 3-byte IANA for the OEM or body that defined the Cipher Suite.
var	00b	<u>Authentication Algorithm Number.</u> A Cipher Suite is only allowed to utilize one Authentication algorithm. See <i>Table 2-18, Authentication Algorithm Numbers</i>
var	01b	<u>Integrity Algorithm Number(s).</u> See <i>Table 2-19, Integrity Algorithm Numbers</i>
var	10b	<u>Confidentiality Algorithm Number(s).</u> See <i>Table 2-20, Confidentiality Algorithm Numbers</i>

2.10.2 Cipher Suite IDs

The following table provides the number ranges and assignments for Cipher Suite IDs. The Cipher Suite ID values are used as a way to identify different Cipher Suites in configuration parameters and IPMI commands.

The OEM IDs do not correspond to a particular Cipher Suite, but are handles that can be used to identify the Cipher Suite on a particular implementation of a BMC. I.e. the OEM Cipher Suite corresponding to “80h” can be different from one BMC to the next. These handles can, however, be used in configuration parameters and commands the same way as the IPMI-defined Cipher Suite IDs.

The *Get Channel Cipher Suites* command will return the algorithms used to form a given Cipher Suite (those numbers can then be used by a remote console in the commands for establishing a session). For OEM defined Cipher Suites, the *Get Channel Cipher Suites* command will also return the IANA for the OEM or body that defined the Cipher Suite.

Table 2-6, Cipher Suite IDs

ID	characteristics	Cipher Suite	Authentication Algorithm	Integrity Algorithm(s)	Confidentiality Algorithm(s)
1	S	(01h, 00h, 00h)	RAKP-HMAC-SHA1	None	None
2	S, A	(01h, 01h, 00h)	RAKP-HMAC-SHA1	HMAC-SHA1-96	None
3	S, A, E	(01h, 01h, 01h)	RAKP-HMAC-SHA1	SHA1 HMAC-SHA1-96	AES-CBC-128
80h-BFh	OEM specified	OEM specified	OEM specified	OEM specified	OEM specified
C0h-FFh	reserved	-	-	-	-

Key:

S = authenticated session setup (correct role, username and password/key required to establish session)

A = authenticated payload data supported.

E = authentication and encrypted payload data supported

2.11 RMCP/ASF Pong Message (Ping Response)

Table 2-7, RMCP Packet Fields for ASF Presence Pong Message (Ping Response)

	Field	size in bytes	Value
UDP Header	Source Port	2	26Fh
	Destination Port	2	from Ping request
	UDP Length	2	per UDP
	UDP Checksum	2	per UDP
RMCP Header	Version	1	6 = RMCP Version 1.0
	Reserved	1	00h
	RMCP Sequence Number	1	FFh for IPMI ^[2]
	Class of Message	1	06h = ASF
ASF Message	IANA Enterprise Number	4	4542 = ASF IANA
	Message Type	1	40h = Presence Pong
	Message Tag	1	from Ping request
	Reserved	1	00h
	Data Length	1	16 (10h)
	IANA Enterprise Number	4	If no OEM-specific capabilities exist, this field contains the ASF IANA (4542) and the OEM-defined field is set to all zeroes (00000000h). Otherwise, this field contains the OEM's IANA Enterprise Number and the OEM-defined field contains the OEM-specific capabilities. For IPMI v2.0 this field can contain the IPMI IANA for 'quick discovery'. If the IANA = IPMI (00001BF2h, 7154 decimal) then the OEM-defined field is used to tell the remote console what level of IPMI over LAN command support is available, provided the 'IPMI supported' bit in the supported entities field (below) is also set. This is recommended for new implementations, but is not mandatory. If the IPMI IANA is not provided, the 'IPMI supported' bit indicates that the remote console must use the <i>Get Channel Authentication Capabilities</i> command to determine the level of IPMI over LAN support.
	OEM-defined	4	This field can contain OEM-defined values; the definition of these values is left to the manufacturer identified by the preceding IANA Enterprise number. For IANA = IPMI (byte 1 returned first): <u>byte 1:</u> [7:2] - reserved [1] - 1b = supports IPMI v2.0/RMCP+ connections. [0] - 1b = supports IPMI v1.5 connections.

		byte 2:4 - reserved (return as 00h)
Supported Entities	1	81h for IPMI [7] 1b = IPMI Supported [6:4] Reserved [3:0] 0001b = ASF Version 1.0
Supported Interactions ^[1]	1	Reserved for future definition by ASF specification, set to 00000000b
Reserved	6	Reserved for future definition by ASF specification, set to 00 00 00 00 00 00h

2.12 AES-CBC-128 Encrypted Payloads

The following table summarizes the contents of the IPMI Payload when AES-CBC encryption is used.

Table 2-8, AES-CBC Encrypted Payload Fields

Field	Size	Sub field	Description
Confidentiality Header	16	Initialization Vector	For the AES algorithm in CBC mode, this field must be a 16-byte random value generated uniquely for each message (packet).
Payload Data	variable	Payload Data	
Confidentiality Trailer	variable	Confidentiality Pad	Added to the Data field to be encrypted (including the Confidentiality Pad Length field) so that they have a length that is a multiple of the block size of algorithm being used. For the AES algorithm, the block size is 16 bytes.
	1	Confidentiality Pad Length	Defines the number of Confidentiality Pad bytes present in the message. For the AES algorithm, this number will range from 0 to 15 bytes. This field is mandatory. If no <i>Confidentiality Pad</i> bytes are required, the <i>Confidentiality Pad Length</i> field is set to 00h. If present, the value of the first byte of <i>Confidentiality Pad</i> shall be one (01h) and all subsequent bytes shall have a monotonically increasing value (e.g., 02h, 03h, 04h, etc). The receiver, as an additional check for proper decryption, shall check the value of each byte of <i>Confidentiality Pad</i> . Some messages may not require padding if the messages already provide the necessary alignment.

2.12.1 Generating the Initialization Vector

The initialization vector (IV) should be unpredictable. In particular, for any given plaintext, it must not be possible to predict what the next IV will be from the last IV. For AES-CBC-128, the IV is recommended to be a 16-byte random number generated by a high quality random number generation process. See *Section 2.25, Random Number Generation*.

2.12.2 Encryption with AES

AES-128 uses a 128-bit Cipher Key. The Cipher Key is the first 128-bits of key “K2”, K2 is generated from the Session Integrity Key (SIK). See *Section 2.24, Generating Additional Keying Material*.

Once the Cipher Key has been generated it is used to encrypt the payload data. The payload data is padded to make it an integral numbers of blocks in length (a block is 16 bytes for AES). The payload is then encrypted one block at a time from the lowest data offset to the highest using Cipher_Key as specified in [AES].

2.12.3 CBC (Cipher Block Chaining)

When CBC is used, before a block of payload data is encrypted it is first exclusive-ORed with the previous ciphertext block (or in the case of the first block, with the initialization vector). For AES, this means that the each 16-byte block of plaintext payload data is exclusive-ORed with the previous 16-bytes of encrypted data before being encrypted. See [MODES] for information on CBC.

For AEC-CBC-128 encrypted payloads under IPMI v2.0 RMCP+, CBC does not span between packets, it only applies to blocks within a packet. Instead, each individual packet is encrypted using a different initialization vector. Thus, packets can be decrypted even if an intermediate block is lost.

2.13 RMCP+ Authentication

RMCP+ session establishment messages are specified under the IPMI Message class for RMCP. The following messages are specified:

1. RSSP Open Session Request
2. RSSP Open Session Response
3. RAKP Message 1
4. RAKP Message 2
5. RAKP Message 3

The Payload Type field identifies which message is being carried in the IPMI Payload portion of the packet.

2.13.1 RSSP Open Session Request

The remote console sends this RSSP message to the managed system to open a protected session. The client responds with an *RSSP Open Session Response* message. Following the *Remote Console Session ID* field, this message contains one or more *Authentication Payload* proposals, one or more *Integrity Payload* proposals, and one or more *Confidentiality Payload* proposals. If the *RSSP Open Session Request* is accepted, the BMC has found a Cipher Suite that matches up with the one or more combinations of the algorithm proposals.

Note: While the format of the *RSSP Open Session Request* incorporates payload fields that can support multiple algorithms of each type, this revision of the specification only requires supporting one (1) algorithm or empty ‘wildcard’ field for each security algorithm payload type in the request.

The format of this message is as follows:

Table 2-9, RSSP Open Session Request

	byte	Data field
IPMI Session Header		<u>Payload Type</u> = RSSP Open Session Request <u>Session Sequence Number</u> = 00_00_00_00h <u>Session ID</u> = 00_00_00_00h
IPMI Payload	1	<u>Message Tag</u> - Selected by remote console.
	2	<u>Requested Maximum Privilege Level (Role)</u> [7:4] - Reserved for future definition by this specification, set to 0h [3:0] - Requested Maximum Privilege Level (Role). 0h = Highest level matching proposed algorithms. BMC will pick the Cipher Suite returned in the <i>RSSP Open Session Response</i> by checking the algorithms proposed in the <i>RSSP Open Session Request</i> against the Cipher Suites available for each privilege level, starting with the “OEM Proprietary level” and progressing to lower privilege levels until a match is found. The resultant match results in an ‘effective’ maximum privilege level for the session. The resultant level is returned in the <i>RSSP Open Session Response</i> . 1h = CALLBACK level 2h = USER level 3h = OPERATOR level 4h = ADMINISTRATOR level 5h = OEM Proprietary level
	3:4	<u>reserved</u> - write as 00_00h
	5:8	<u>Remote Console Session ID</u> . Selected by the remote console to identify packets that are received for the given session by the remote console
	9:13	<u>Authentication Payload</u> . Identifies the authentication type that the managed system wants to use for the session. <u>byte 1</u> - Payload Type 00h = authentication algorithm <u>byte 2:3</u> - reserved = 0000h <u>byte 4</u> - Payload Length in bytes (1-based). 00h = Null field (“wildcard”). BMC picks algorithm based on Requested Maximum Privilege Level and that matches with the proposed Integrity and Confidentiality payloads. If the Requested Maximum Privilege Level is ‘unspecified’ the BMC picks algorithm based on the Integrity and Confidentiality algorithm proposals starting from the highest privilege level until a match is found. <u>byte 5 - Authentication Algorithm</u> [7:6] - <u>reserved</u> [5:0] - Authentication Algorithm (See <i>Table 2-18, Authentication Algorithm Numbers</i>) <u>byte 6:8</u> - <u>reserved</u>
	14:18	<u>Integrity Payload</u> . Identifies the integrity type that the managed system wants to use for the session. <u>byte 1</u> - Payload Type 01h = integrity algorithm

	<u>byte 2:3</u> - reserved = 0000h <u>byte 4</u> - Payload Length in bytes(1-based) 00h = Null field ("wildcard"). BMC picks algorithm based on Requested Maximum Privilege Level and that matches with the proposed Authentication and Confidentiality payloads. If the Requested Maximum Privilege Level is 'unspecified' the BMC picks algorithm based on the Authentication and Confidentiality algorithm proposals starting from the highest privilege level until a match is found. <u>byte 5</u> - Integrity Algorithm [7:6] - reserved [5:0] - Integrity Algorithm (See <i>Table 2-19, Integrity Algorithm Numbers</i>) <u>byte 6:8</u> - reserved
19:23	<u>Confidentiality Payload</u> . Defined confidentiality algorithms are: <u>byte 1</u> - Payload Type 02h = confidentiality algorithm <u>byte 2:3</u> - reserved = 0000h <u>byte 4</u> - Payload Length in bytes (1-based) 00h = Null field ("wildcard"). BMC picks algorithm based on Requested Maximum Privilege Level and that matches with the proposed Authentication and Integrity payloads. If the Requested Maximum Privilege Level is 'unspecified' the BMC picks algorithm based on the Authentication and Integrity algorithm proposals starting from the highest privilege level until a match is found. <u>byte 5</u> - Confidentiality Algorithm [7:6] - reserved [5:0] - Confidentiality Algorithm (See <i>Table 2-20, Confidentiality Algorithm Numbers</i>) <u>byte 6:8</u> - reserved

2.13.2 RSSP Open Session Response

A managed system sends this RSSP message to the management console in response to an *RSSP Open Session Request* message. Following the *Status Code*, *Mgt Console* and *Managed System Session ID* fields, this message contains a single *Authentication* payload, a single *Integrity* payload, and a single *Confidentiality* payload. These payloads represent the proposals that the managed system selected from the list offered by the management console.

The format of this message's *Data* section is as follows:

Table 2-10, RSSP Open Session Response

	byte	data field
IPMI Session Header		<u>Payload Type</u> = RSSP Open Session Response <u>Session Sequence Number</u> = 00_00_00_00h <u>Session ID</u> = 00_00_00_00h
IPMI Payload	1	<u>Message Tag</u> - Selected by remote console.
	2	<u>RSSP Status Code</u> - Identifies the status of the previous message. If the previous message generated an error, then only the Status Code, Reserved, and Remote Console Session ID fields are returned. See <i>Table 1-15, RSSP and RAKP Message Status Codes</i> . The session is then immediately closed at the BMC.

3	<u>Maximum Privilege Level (Role)</u> - Indicates the Maximum Privilege Level allowed for the session based on the security algorithms that were proposed in the <i>RSSP Open Session Request</i> . [7:4] - Reserved for future definition by this specification, set to 0h [3:0] - Requested Maximum Privilege Level (Role). 0h = unspecified (returned with error completion code). 1h = CALLBACK level 2h = USER level 3h = OPERATOR level 4h = ADMINISTRATOR level 5h = OEM Proprietary level
4	<u>reserved</u> - write as 00h
5:8	<u>Remote Console Session ID</u> The Remote Console Session ID specified by <i>RSSP Open Session Request</i> message associated with this response.
9:12	<u>Managed System Session ID</u> The Session ID selected by the Managed System for this new session. The Bypass Session ID (All 0's) is not valid in this context.
13:20	<u>Authentication Payload</u> This payload defines the authentication algorithm proposal selected by the Managed System to be used for this session (see <i>Table 2-9, RSSP Open Session Request</i> for the definition of this payload). A single algorithm will be returned. The 'Null field' is not allowed.
21:28	<u>Integrity Payload</u> This payload defines the integrity algorithm proposal selected by the Managed System to be used for this session (see <i>Table 2-9, RSSP Open Session Request</i> for the definition of this payload). A single algorithm will be returned. The 'Null field' is not allowed.
29:36	<u>Confidentiality Payload</u> This payload defines the confidentiality algorithm proposal selected by the Managed System to be used for this session (see <i>Table 2-9, RSSP Open Session Request</i> for the definition of this payload). A single algorithm will be returned. The 'Null field' is not allowed.

2.14 Close Session Command

This IPMI command is used to immediately terminate a session in progress. It is typically used to close the session that the user is communicating over, but it can also be used to other terminate sessions in progress (provided that the user is operating at the appropriate privilege level, or the command is executed over a non-authenticated channel - e.g. the system interface).

Table 2-11, Close Session Command

	byte	data field
Request Data	1:4	Session ID
Response Data	1	Completion Code 87h = invalid session ID in request

2.15 Session Termination

The following actions can terminate a session:

- The Close Session command
- Session Inactivity Timeout (Per IPMI v1.5 section 6.11.13, Session Inactivity Timeouts)

Terminating a session causes any payloads that were activated under that session to be automatically deactivated. Terminating one session will not cause other sessions to

terminate. If multiple sessions are opened by a remote console they will need to be terminated individually.

2.16 RAKP Messages

RAKP Messages are transferred in the payload portion of an IPMI over LAN packet with a Format field set to “RMCP+”. The Payload Type field indicates which RAKP message is held in the IPMI Payload portion of the packet.

2.16.1 RAKP Message 1

A remote console sends this RAKP message to the managed system to begin the session authentication process. The remote console selects a *Remote Console Random Number*, a *Maximum Requested Privilege Level (Role)*, and an optional *User Name* and sends them to the managed system along with the *Managed System Session ID* specified by the client on the previous *RSSP Open Session Response*.

Upon receiving *RAKP Message 1*, the managed system verifies that the message contains an active *Managed System Session ID* and that a session can be created using the given user information and security algorithm proposal. The managed system responds with an *RAKP Message 2*.

FYI: RAKP Message 1 corresponds roughly to an IPMI v1.5 *Get Session Challenge* request message. RAKP Message 2 corresponds roughly to an IPMI *Get Session Challenge* response message.

The format of the IPMI Session Header, Payload, and Session Trailer for the *RAKP Message 1* is shown in the following table.

Table 2-12, RAKP Message 1

	byte	data field
IPMI Session Header		<u>Payload Type</u> = RAKP Message 1 <u>Session Sequence Number</u> = 00_00_00_00h <u>Session ID</u> = 00_00_00_00h
IPMI Payload	1	<u>Message Tag</u> - Selected by remote console.
	2:4	<u>reserved</u> - write as 00_00h
	5:8	<u>Managed System Session ID</u> The Managed System's Session ID for this session, returned by the Managed System on the previous <i>RSSP Open Session Response</i> message.
	9:24	<u>Remote Console Random Number</u> Random number selected by the Remote Console

25	<u>Requested Maximum Privilege Level (Role)</u> [7:5] - Reserved for future definition by this specification, set to 000b [4] - 0b = <u>Username/Privilege lookup</u>. Both the Requested Privilege Level and User Name are used to look up password/key. The BMC will search the user entries starting with USER ID 1 and use the first entry that matches the specified user name and has a Maximum Privilege Level that matches the Requested Privilege Level. This can be used in combination with 'null' user names to enable a password that is just based on the requested privilege level. 1b = <u>Name-only lookup</u>. User Name alone is used to look up password/key. Privilege Level field acts as a 'Maximum Requested Privilege Level' ala IPMI v1.5. The rules for privilege level handling are summarized as follows: • If the Requested Privilege Level is greater than the privilege limit for the channel/user, the user will be allowed to connect but will be restricted to the channel/user privilege limit that was configured for the user. • If the Requested Privilege Level is less than the channel/user privilege limit, the user will be allowed to connect and Request Privilege Level will become the effective privilege limit for the user. I.e. the user will not be able to raise their privilege level higher than the Requested Privilege Level. [3:0] - Requested Maximum Privilege Level 0h = reserved 1h = CALLBACK level 2h = USER level 3h = OPERATOR level 4h = ADMINISTRATOR level 5h = OEM Proprietary level
26:27	reserved - write as 0000h
28	<u>User Name Length</u> 00h No name present 01h-10h Name length 11h-FFh Reserved for future definition by this specification
29:44	<u>User Name</u> ASCII character Name that the user at the Remote Console wishes to assume for this session. No NULL characters (00h) are allowed in the name. Sixteen-bytes, max.

2.16.2 RAKP Message 2

The managed system sends this RAKP message to a remote console in response to the receipt of an *RAKP Message 1*. Once *RAKP Message 1* has been validated the managed system selects a *Managed System Random Number* and computes a *Key Exchange Authentication Code* over the values specified by the RAKP algorithm. The managed system sends those values along with the *Managed System Globally Unique ID (GUID)* and the *Remote Console Session ID* (sent by the console on the previous *RSSP Open Session Request*) to the remote console.

Upon receiving *RAKP Message 2*, the remote console verifies that the *Remote Console Session ID* is active and that the *Managed System GUID* matches the managed system that the remote console has associated with the session. The remote console then validates the *AuthCode (Integrity Check Value)* and responds with an *RAKP Message 3*.

The format of an *RAKP Message 2* message's *Data* section is as follows:

Table 2-13, RAKP Message 2

	byte	data field
IPMI Session Header		<u>Payload Type</u> = RAKP Message 2 <u>Session Sequence Number</u> = 00_00_00_00h <u>Session ID</u> = 00_00_00_00h ← TBD :need to verify whether this is 0 or not.
IPMI Payload	1	<u>Message Tag</u> - Selected by remote console.
	2	<u>RSSP Status Code</u> - Identifies the status of the previous message. If the previous message generated an error, then only the Completion Code, Reserved, and Remote Console Session ID fields are returned. If the Remote Console Session ID field is indeterminate (as would be the case if the Managed System Session ID in RAKP Message 1 were invalid) then the Remote Console Session ID field will be set to all zeros. On error, the remote console can attempt to correct the error and send a new RAKP Message 1. Note that the remote console must change the Message Tag value to ensure the BMC sees the message as a new message and not as a retry. See Table 1-15, <i>RSSP and RAKP Message Status Codes</i> for the status codes defined for this message.
	3:4	Reserved - write as 00_00h.
	5:8	Remote Console Session ID - The Remote Console Session ID specified by the <i>RSSP Open Session Request</i> message associated with this response.
	9:24	Managed System Random Number - Random number selected by the managed system.
	25:40	Managed System GUID - The Globally Unique ID (GUID) of the Managed System. This value is specified by the client system's SMBIOS implementation.
	41:N	Key Exchange Authentication Code An integrity check value over the relevant items specified by the RAKP algorithm for RAKP Message 2. The size of this field depends on the specific Authentication Algorithm (e.g. for RAKP-HMAC-SHA1 the Authentication Algorithm would be HMAC using SHA1) that was identified in the <i>RSSP Open Session Response</i> .

2.16.3 RAKP Message 3

A remote console sends this RAKP message to a managed system in response to the receipt of an *RAKP Message 2*. Once it validates *RAKP Message 2*, the remote console creates a *Session Integrity Key* using the values specified by the RAKP algorithm. The remote console then computes an *Integrity Check Value* over the values specified by the RAKP algorithm, and sends that along with the *Managed System Session ID* (sent by the managed system on the previous *RSSP Open Session Response* message) to the managed system.

After receiving *RAKP Message 3*, the managed system verifies that the *Managed System Session ID* is active and then validates the *Integrity Check Value*. If the *Integrity Check Value* is valid, the managed system creates a *Session Integrity Key* using the values specified by the RAKP algorithm. With the shared *Session Integrity Key* in place,

integrity protected messages can now be exchanged between the remote console and the managed system.

The format of an *RAKP Message 3* message's *Data* section is as follows:

Table 2-14, RAKP Message 3

byte	data field
IPMI Session Header	<u>Payload Type</u> = RAKP Message 3 <u>Session Sequence Number</u> = 00_00_00_00h <u>Session ID</u> = 00_00_00_00h ← TBD :need to verify whether this is 0 or not.
1	<u>Message Tag</u> - Selected by remote console.
2	<u>RSSP Status Code</u> Identifies the status of the previous message. If the previous message generated an error, then only the Completion Code, Reserved, and Managed System Session ID fields are returned. If the BMC receives an error from the remote console, it will immediately close the session if the Key Exchange Authentication code (below) is valid. See <i>Table 1-15, RSSP and RAKP Message Status Codes</i> for the status codes defined for this message.
3:4	Reserved - write as 00_00h.
5:8	<u>Managed System Session ID</u> The Managed System's Session ID for this session, returned by the managed system on the previous <i>RSSP Open Session Response</i> message.
9:N	<u>Key Exchange Authentication Code</u> An integrity check value over the relevant items specified by the RAKP algorithm for Message 3. The size of this field depends on the specific Authentication Algorithm that was selected when the session was created.

2.16.4 RAKP Message 4

A managed client sends this RAKP message to a management console in response to the receipt of an *RAKP Message 3*. Once *RAKP Message 3* has been validated, the managed client computes an *Integrity Check Value* over the values specified by the RAKP algorithm. The managed client then sends the *Mgt Console Session ID* and the *Integrity Check Value* to the management console.

Upon receiving *RAKP Message 4*, the management console verifies that the *Mgt Console Session ID* is active and then validates the *Integrity Check Value*.

The format of an *RAKP Message 4* message's *Data* section is as follows:

Table 1-14, RAKP Message 4

byte	data field
IPMI Session Header	<u>Payload Type</u> = RAKP Message 4 <u>Session Sequence Number</u> = 00_00_00_00h <u>Session ID</u> = 00_00_00_00h ← TBD :need to verify whether this is 0 or not.
1	<u>Message Tag</u> - Selected by remote console.
2	<u>RSSP Status Code</u> - Identifies the status of the previous message. If the previous message generated an error, then only the Status Code, Reserved, and Management Console Session ID fields are returned. See 2.1.3.6.1 for the status codes defined for this message.
3:4	<u>Reserved</u> - Reserved for future definition by this specification set to 000000h.
5:8	<u>Mgt Console Session ID</u> The Mgt Console Session ID specified by the <i>RSSP Open Session Request (83h)</i> message associated with this response.
9:N	<u>Integrity Check Value</u> An integrity check value over the relevant items specified by the RAKP algorithm for Message 4. The size of this field depends on the specific algorithm that was selected when the session was created.

2.17 RSSP and RAKP Message Status Codes

The table below lists the status codes for specific RSSP and RAKP messages.

Table 1-15, RSSP and RAKP Message Status Codes

Status Code	Description	Message			
		RSSP Open Session Response	RAKP Msg 2	RAKP Msg 3	RAKP Msg 4
00h	No errors	X	X	X	X
01h	Insufficient resources to create a session	X			
02h	Invalid session ID	X	X	X	X
03h	Invalid payload type	X			
04h	Invalid authentication algorithm	X			
05h	Invalid integrity algorithm	X			
06h	No matching authentication payload	X			
07h	No matching integrity payload	X			
08h	Inactive session ID		X	X	X
09h	Invalid role		X		
0Ah	Unauthorized role		X		
0Bh	Insufficient resources to create a session at the requested role		X		
0Ch	Invalid name length		X		
0Dh	Unauthorized name		X		
0Eh	Unauthorized GUID			X	
0Fh	Invalid integrity check value			X	X
10h	Invalid confidentiality algorithm	X			
11h	No Cipher Suite match with proposed security algorithms	X			
12h-FFh	Reserved for future definition by this specification				

2.18 IPMI v2.0 RMCP+ Payload Types

The Payload Type field in the IPMI v2.0/RMCP+ packet carries a *Payload Type Number* that identifies what type of payload field is being carried in that particular packet.

The *Payload Type Numbers* (also referred to as *payload types*) are classified into three main categories: Standard Payload Types - used to identify payloads that are specified by the IPMI specifications, Session Setup Payload Types - used to identify payloads that are for session setup messages specified by the IPMI specifications, and OEM Payload Types that are used to identify payloads that are specified by a given OEM.

The following table lists the assignment and ranges of the Payload Type Numbers. The complete identification of an OEM Payload is given by the combination of a three-byte IANA ID for the OEM plus a two-byte OEM Payload ID that is assigned and defined by the given OEM. These can either be carried explicitly in each packet (adding six bytes of overhead) or an application can elect to use an OEM Payload Type Handle. The OEM Payload Type Handle in the Payload Type provides a value that represents a particular OEM IANA and OEM Payload ID on a system. The *Get Channel Payload Support* command is used discover what OEM Payloads (if any) are used on the managed system, and to associate the OEM Payload Type number with the OEM IANA ID and Payload ID.

OEM Payload Handle assignments can vary from system to system. For example, “OEM0” on one system may not correspond to the same type of OEM Payload as “OEM0” on another system. Software that uses OEM Payload Handles must not assume that a given OEM Payload Handle number will correspond to a particular OEM IANA and Payload ID combination across multiple systems. Software must use the *Get Channel Payload Support* command to discover the relationship.

Associated with each payload type is a format version number that provides information on the backward compatibility with different versions of the payload type. See *Section 3.9, Get Channel Payload Version Command*, for more information.

Table 2-17, Payload Type Numbers

number*	type	major format version	minor format version
<i>Standard Payload Types</i>			
0h	IPMI Message	1h	0h
1h	SOL (serial over LAN)	1h	0h
2h	OEM Explicit (This payload type indicates that the packet includes explicit OEM IANA and OEM Payload ID fields, rather than using an OEM Payload Type Handle to identify the type of payload. When used, this option adds 6 bytes to the overhead of the packet.)	OEM specified according to OEM IANA and OEM Payload ID.	OEM specified according to OEM IANA and OEM Payload ID.
<i>Session Setup Payload Types</i>			
10h	RSSP Open Session Request	1h	0h
11h	RSSP Open Session Response	1h	0h
12h	RAKP Message 1	1h	0h
13h	RAKP Message 2	1h	0h
14h	RAKP Message 3	1h	0h
15h	RAKP Message 4	1h	0h
<i>OEM Payload Type Handles</i>			
20h-27h	OEM payloads OEM0 through OEM7, respectively.	OEM specified according to OEM IANA and OEM Payload ID.	OEM specified according to OEM IANA and OEM Payload ID.
all other	reserved		

* The payload type number is a 6-bits (00h-3Fh).

2.19 Session Sequence Numbers

The session sequence number is a 32-bit, unsigned, value. The session sequence number is used for protecting against replay attacks. The session sequence number is *not* used for matching IPMI requests and responses or tracking lost packets. The IPMI Sequence (Seq) field, or similar field in the particular payload is used for that purpose. The sender of the packet increments the session sequence number for every packet that gets transmitted even if the payload of the content is a ‘retry’. Session Sequence Numbers are generated and tracked on a per-session basis. I.e. there are separate sets of sequence numbers and sequence number handling for each session.

Sequence numbers only apply to packets that are transmitted within the context of an IPMI session. Certain IPMI commands and protocol messages are accepted ‘outside of a session’. For these packets, the sequence number fields are always set to 0000_0000h.

2.20 IPMI v1.5 Session Sequence Number Handling

For IPMI v1.5 sessions, there are just two Session Sequence Numbers associated with a given session: the *Inbound* Session Sequence Number and the *Outbound* Session Sequence Numbers. (The inbound and outbound directions are defined with respect to the BMC. Inbound messages are from the remote console to the BMC, while outbound messages are from the BMC to the remote console.) The inbound and outbound sequence numbers are updated and tracked independently. I.e. since the number of incoming packets and outgoing packets will typically vary, the inbound and outbound sequence numbers will not stay in lock step with one another.

The BMC and the remote console independently select the starting session sequence number for the messages they receive. Typically, this is done using a random number in order to further reduce the likelihood of a playback attack. The remote console sets the starting values for the outbound session sequence number when it sends the first *Activate Session* command for an authenticated session. The *Activate Session* response is the first authenticated outbound (BMC to remote console) message. This response message uses the initial outbound session sequence number value that the remote console delivered in the prior *Activate Session* command request. The BMC must increment the outbound session sequence number by one (1) for each subsequent outbound message from the BMC.

The first authenticated inbound message uses the initial inbound (remote console to BMC) session sequence number value that was returned by the BMC in the response to the *Activate Session* command. The remote console must increment the inbound session sequence number by one (1) for each subsequent message it sends to the BMC.

2.20.1 IPMI v1.5 Inbound Session Sequence Number Tracking and Handling

Session sequence numbers are tracked on a per-session basis. At a minimum, the BMC is required to track that the inbound sequence number is increasing, and to silently discard the packet if the sequence number is **eight** counts or more from than the last value received. (An implementation is allowed to contain a proprietary configuration option that enables a larger sequence number difference, as long as the standard of +eight can be restored.)

An implementation can elect to terminate the session if it receives a number of sequence numbers that are more than eight counts from the last value received.

Valid packets (packets with good data integrity checks and signature) to a given session that have the same inbound sequence number as an earlier packet are considered to be duplicate packets and are silently discarded (even if the message content is different).

2.20.2 IPMI v1.5 Out-of-order Packet Handling

In order to avoid closing a session because a packet was received out-of-order, the BMC must implement one of two options:

Option 1: Advancing eight-count (or greater) window. Recommended. Track which packets have been received that have sequence numbers up to eight counts less than the highest last received sequence number, tracking which of the prior eight sequence numbers have been received. Also accept packet with sequence numbers that are up to eight counts greater than the last received sequence number, and set that number as the new value for the highest sequence number received. This option is illustrated in [Appendix A - Previous Sequence Number Tracking](#).

Option 2: Drop any packets with sequence numbers that are lower than the last valid value received. While simpler than option 1, this option is not recommended except for resource-constrained implementations due to the fact that any out-of-order packets will require the remote console to timeout and retransmit.

Sequence number wrap-around must be taken into account. When a sequence number advances from FFFF_FFFFh to 0000_0000h, the an eight count window will range from FFFF_FFF9h to 0000_0000h. The next window would go from FFFF_FFFAh to 0000_00001h, etc.

When dropping packets because of sequence number, any packet with an illegal, duplicate, or out-of-range sequence can be dropped without having to verify the packet integrity data (AuthCode) signature first. When accepting packets, the BMC must apply any packet integrity and authentication code checks before accepting the packet's sequence number.

2.20.3 IPMI v1.5 Outbound Session Sequence Number Tracking and Handling

The remote console is required to handle outbound session sequence number tracking in the same manner as the BMC handles the inbound session sequence number, except that Option 2 (above) should not be used as a means of handling out-of-order packets.

2.21 IPMI v2.0 RMCP+ Session Sequence Number Handling

For IPMI v2.0 RMCP+ sessions, there are two sets of Session Sequence Numbers for a given session. One set of inbound and outbound sequence numbers is used for authenticated (signed) packets, and the other set is used for unauthenticated packets. The inbound and outbound sequence numbers for authenticated packets are updated and tracked independently from the inbound and outbound sequence numbers for unauthenticated packets.

IPMI v2.0 RMCP+ Session Sequence Numbers are used for rejecting packets that may have been duplicated by the network or intentionally replayed.

The individual Session Sequence Numbers is are initialized to zero whenever a session is created and incremented by one at the start of outbound processing for a given packet (i.e. the first transmitted packet has a '1' as the sequence number, not 0). Session Sequence numbers are incremented for every packet that is transmitted by a given sender, regardless of whether the payload for the packet is a 'retry' or not.

For IPMI v2.0 RMCP+ packets, sequence numbers don't wrap. If the sequence number reaches FFFF_FFFFh the session is automatically closed and must be re-established by the remote console.

When dropping packets because of sequence number, any packet with an illegal, duplicate, or out-of-range sequence can be dropped without having to verify the packet integrity data (AuthCode) signature first. When accepting packets, the BMC must apply

any packet integrity and authentication code checks before accepting the packet's sequence number.

2.21.1 IPMI v2.0 RMCP+ Sliding Window

IPMI v2.0 RMCP+ uses a 'sliding window' for tracking sequence numbers for received packets. This sliding window is used for rejecting packets that have sequence numbers that are significantly out-of-range with respect to the sequence number for the most recently accepted packet while allowing a number of out-of-order packets to be accepted.

In order for a packet to be accepted by the BMC, its sequence number must fall within a 32 count sliding window, where packets will be accepted if they are within plus 15 or minus 16 counts of the highest sequence number that was previously accepted, and they are not duplicates of any previously received sequence numbers.

2.22 Authentication, Integrity, and Encryption Algorithm Numbers

The Authentication Algorithm Number specifies the type of authentication "handshake" process that is used and identifies any particular variations of hashing or signature algorithm that is used as part of the process. For example, RAKP-HMAC-SHA1 specifies the use of RAKP messages for the key exchange portion of establishing the session, and that HMAC-SHA1 is used to create the Key Exchange Authentication Code that is a parameter in RAKP Message 2 and RAKP Message 3.

Table 2-18, Authentication Algorithm Numbers

number*	type	Mandatory / Optional ¹
00h	none	M
01h	RAKP-HMAC-SHA1	M
02h	RAKP-HMAC-MD5-128	O
03h	RAKP-MD5-128	O
C0h-FFh	OEM	O
all other	reserved	-

* The number range is limited to six (6) bits (00h-3Fh)

The Integrity Algorithm Number specifies the algorithm used to generate the contents for the AuthCode "signature" field that accompanies authenticated IPMI v2.0/RMCP+ messages once the session has been established. Unless otherwise specified, the algorithm applies to the packet data starting with the AuthType/Format field up to and including the field that immediately precedes the AuthCode field itself. The AuthCode field is a common part of the IPMI

Table 2-19, Integrity Algorithm Numbers

¹ Mandatory/Optional is with respect to BMC support. It is recommended that remote consoles support all specified algorithms in order to support maximum number of BMC implementations.

number*	type	Mandatory / Optional ¹
00h	none	M
01h	HMAC-SHA1-96	M
02h	HMAC-MD5-128	O
03h	MD5-128	O
C0h-FFh	OEM	O
all other	reserved	-

* The number range is limited to six (6) bits (00h-3Fh)

The Confidentiality Algorithm Number specifies the encryption/decryption algorithm field that is used for encrypted payload data under the session. The ‘encrypted’ bit in the payload type field being set identifies packets with payloads that include data that is encrypted per this specification. When payload data is encrypted, there may be additional “Confidentiality Header” and/or “Confidentiality Trailer” fields that are included within the payload. The size and definition of those fields is specific to the particular confidentiality algorithm.

Table 2-20, Confidentiality Algorithm Numbers

number*	type	Mandatory / Optional ¹
00h	none	M
01h	AES-CBC-128 (See Section 2.12, AES-CBC-128 Encrypted Payloads, for more information)	M
02h	RC4-128-MD5	O
C0h-FFh	OEM	O
all other	reserved	-

* The number range is limited to six (6) bits (00h-3Fh)

2.23 RSSP Authenticated Key-Exchange Protocol (RAKP)

RSSP can support a number of different authentication and key exchange protocols during its *Creation* phase. For this specification, the mandatory-to-implement authentication and key exchange protocol is the RSSP Authenticated Key-Exchange Protocol (RAKP). RAKP (defined below) was developed based on the Authenticated Key Exchange Protocol (AKEP) defined by Bellare and Rogaway in [BR1].

RAKP uses pre-shared symmetric keys and HMAC-based integrity algorithms to mutually authenticate a remote console to a given managed system and to generate pairwise unique symmetric keying material that can be used with a number of integrity and confidentiality algorithms to provide protection for RMCP messages. For this specification, RAKP shall use the HMAC-SHA1 integrity algorithm defined in [RFC2104] and the HMAC-SHA1-96 integrity algorithm defined in [RFC2404].

RAKP also supports the concept of remote console user “roles” and optionally “usernames” (e.g. operator “x” or administrator “y”), which are established by RAKP when a session is created.

Examples of behavior that can be controlled include the roles that the managed system can use to establish sessions (e.g. operator-only sessions) and the roles and names

(optional) allowed to execute each RMCP message the managed system might receive during a given session.

Before a given managed system's RMCP implementation can become operational, it must be configured with various RMCP-related parameters. This includes installing user passwords (keys) and usernames, setting up access rights for the individual users, and configuring which Cipher Suites are used for authenticated and/or encrypted transfers with the managed system.

The managed system can be configured with keys for each username, or 'null' usernames can be used, in which case the key is associated solely with a given privilege level (role). The different user keys are specified using the notation **K_{UID}**, where UID represents the User ID number that is used in the user-specific configuration commands in IPMI.

The user keys are 'shared secrets' between the BMC and the remote console(s). RAKP/RSSP does not include a secure, confidential mechanism for installing and distributing user keys between BMCs and remote consoles. The installation and distribution of user keys can typically be accomplished with a software utility that uses OS-provided mechanisms for the secure transfer of keys. If authentication and encryption are available, an 'Admin' level user can use IPMI commands such as *Set User Password* for remotely updating and configuring user key and privilege information over an authenticated and confidential session to the BMC.

An additional key, **K_G**, is used for key generation operations. The scope of these keys (shared by multiple managed systems and the remote console or pair-wise unique for each managed system and the remote console) is a local policy issue that is determined by the equipment owner at the time of installation.

Once this and other necessary RMCP-related data is installed in the managed system and the managed system is initialized, the remote console can initiate sessions with the managed system. Following the exchange of RMCP Presence Ping/Pong and RSSP Open Session Request/Response messages (exchanging session IDs and selecting RAKP for use), the remote console starts the RAKP protocol. First, the remote console selects a random number, **R_M**, a requested role, **Role_M**, a user name length, **ULength_M**, a user name (optional – denoted by < > below), **UName_M**, and the managed system's session ID, **SID_C**, and sends them to the managed system as Message 1.

Message 1: Remote Console —► Managed System

SID_C, R_M, Role_M, ULength_M, < UName_M >

After receiving Message 1, the managed system verifies that the value **SID_C** is active and that a session can be created using **Role_M**, **ULength_M**, and (optional), **UName_M** for the given selections for security algorithms.

If the request is valid, the managed system then selects a random number, **R_C**, and sends

to the remote console as Message 2 the values **SID_M**, **R_C**, and **GUID_C** as well as the HMAC per [RFC2404] of the values (**SID_M**, **SID_C**, **R_M**, **R_C**, **GUID_C**, **Role_M**, **ULength_M**, < **UName_M** >) generated using key **K_[UID]** selected by the given username, **UName_M**, and role, **Role_M**.

Message 2: Managed System —► Remote Console

SID_M, **R_C**, **GUID_C**,
**HMAC_{K_[UID]}(SID_M, SID_C, R_M, R_C, GUID_C, Role_M, ULength_M, <
UName_M >)**

Where:

SID_M = Remote_Console_Session_ID
SID_C = Managed_System_Session_ID
R_M = Remote Console Random_Number
R_C = Managed System Random Number
GUID_C = Managed_System_GUID
Role_M = Requested Privilege Level (Role) (this is the entire byte holding the Requested Privilege Level field)
ULength_M = User Name Length byte
UName_M = User Name bytes (if present)

After receiving Message 2, the remote console verifies that the value **SID_M** is active and that **GUID_C** matches the managed system that the remote console is expecting to communicate with. The remote console then validates the HMAC. If the HMAC is valid, the remote console creates the Session Integrity Key (SIK) by generating an HMAC per [RFC2104] of the concatenation of **R_M**, **R_C**, **Role_M**, **ULength_M**, and (optional) **UName_M** using key **K_G** (note – no truncation).

SIK = HMAC_{K_G}(R_M | R_C | Role_M | ULength_M | < UName_M >)

Then the remote console sends to the managed system as Message 3 the value **SID_C** and the HMAC per [RFC2404] of the values (**R_C**, **SID_M**, **Role_M**, **ULength_M**, < **UName_M** >) generated using key **K_[UID]** selected by the username, **UName_M**, and role **Role_M**.

Message 3: Remote Console —► Managed System

SID_C, **HMAC_{K_[UID]}(R_C, SID_M, Role_M, ULength_M, < UName_M >)**

After receiving Message 3, the managed system verifies that the value **SID_C** is active and then validates the HMAC. If the HMAC is valid, the managed system creates the SIK by generating an HMAC per [RFC2104] of the concatenation of **R_M**, **R_C**, **Role_M**, **ULength_M**, and (optional) **UName_M** using key **K_G** (note – no truncation).

$$\text{SIK} = \text{HMAC}_{\text{KG}} (\text{R}_M \mid \text{R}_C \mid \text{Role}_M \mid \text{ULength}_M \mid \langle \text{UName}_M \rangle)$$

The managed system then sends to the management console as Message 4 the values **SID_M**, and the HMAC per [RFC2404] of the values (**R_M**, **SID_C**, **GUID_C**) generated using key **SIK**. The managed system then transitions into the *Message Transfer* session state,

Message 4: Managed System —► Mgt Console

$$\text{SID}_M, \text{HMAC}_{\text{SIK}} (\text{R}_M, \text{SID}_C, \text{GUID}_C)$$

After receiving Message 4, the management console verifies that the value **SID_M** is active and then validates the HMAC. If the HMAC is valid, the management console has verification that mutual authentication with the managed system was successful and that the same pair-wise unique SIK was successfully generated on both ends of the connection. The management console then transitions into the *Message Transfer* session state (the session is now active and, if authentication or authentication/encryption have been enabled, the transfer of authenticated and authenticated/encrypted payloads can commence).

The same RAKP steps are followed for session establishment even if the Cipher Suite indicates that there are no integrity or encryption algorithms required for the session.

2.24 Generating Additional Keying Material

Because this specification supports both integrity and confidentiality services for a session, RSP needs more keying material than can be provided by SIK alone. As a result, all keying material for the RSP integrity and confidentiality algorithms will be generated by processing a pre-defined set of constants using HMAC per [RFC2104], keyed by SIK.

$$\begin{aligned} \text{K}_1 &= \text{HMAC}_{\text{SIK}} (\text{const 1}) \\ \text{K}_2 &= \text{HMAC}_{\text{SIK}} (\text{const 2}) \\ \text{K}_3 &= \text{HMAC}_{\text{SIK}} (\text{const 3}) \end{aligned}$$

These constants are constructed using a hexadecimal octet value repeated up to the HMAC block size in length starting with the constant 01h. This mechanism can be used to derive up to 255 HMAC-block-length pieces of keying material from a single SIK. For the mandatory-to-implement integrity and confidentiality algorithms defined in this specification, processing the first two (2) constants will generate the require amount of keying material.

Const 1	=	0x01010101010101010101010101010101	01010101010101010101010101010101
Const 2	=	0x02020202020202020202020202020202	02020202020202020202020202020202
Const 3	=	0x03030303030303030303030303030303	03030303030303030303030303030303
		.	
		.	
Const 255	=	0xFFFFFFFFFFFFFFFFFFFFFFFF	FFFFFFFFFFFFFFFFFFFFFFFF

2.25 Random Number Generation

Algorithms for authentication (data integrity) and confidentiality (for Initialization Vectors) depend on "quality" random numbers for their security. Quality in this context means that the numbers must be random in a cryptographic sense (i.e., they must be genuinely unpredictable). To ensure that a baseline-level of quality random numbers are provided for remote consoles and managed systems, this specification recommends the following algorithm be used for random number generation for use in authentication and data integrity algorithms if no other higher-quality source of random numbers is available (e.g., a hardware random number generator).

2.25.1 Random Number Key

Each BMC is configured with a 160-bit key, $\mathbf{K_R}$, which is unique for each managed system. The value of this key cannot be reused during the lifetime of $\mathbf{K_{[UID]}}$ and $\mathbf{K_G}$. The *Set Security Keys* command can be used for setting this key. Note that to avoid the possibility of run-time software changing this key, the BMC includes an option for the key to be locked so that run-time software cannot change the key.

2.25.2 Random Number Generator Counters

The managed system maintains two (2) 32 bit counters, $\mathbf{C_P}$ and $\mathbf{C_Q}$. $\mathbf{C_P}$ is used to count the number of device power cycles and its value is saved in non-volatile storage. $\mathbf{C_Q}$ is used to count the number of random number generation requests per power cycle and is volatile. Whenever a new $\mathbf{K_R}$ is installed in non-volatile storage, the counters are reset to zero (0). Once initialized, $\mathbf{C_P}$ is incremented by one (1) after each power cycle and its new value is again saved in non-volatile storage. $\mathbf{C_Q}$ is reset to zero (0) on each power cycle (i.e., its value is not saved across power cycles).

2.25.3 Random Number Generator Operation

The managed system creates a random number by generating an HMAC per [RFC2104] of the concatenation of $\mathbf{C_P}$ and $\mathbf{C_Q}$ using key $\mathbf{K_R}$ using the SHA1 hash function. The output of this generator is a 160-bit pseudo-random number. Uses that require fewer bits can draw the required number of bits from the 160-bit value. Since the value is random, it shouldn't matter which bits are used. Most implementations will simply take either the most-significant 'N' bits or least-significant, whichever is most convenient.

$$\text{Random Number} = \text{HMAC}_{\mathbf{K_R}} (\mathbf{C_P} \mid \mathbf{C_Q})$$

$\mathbf{C_Q}$ is incremented by one (1) after each random number generation request. After each power cycle, the value of $\mathbf{C_Q}$ is reset to zero (0) (i.e., its value is not saved across power cycles). If during a given power cycle, $\mathbf{C_Q}$ rolls-over to zero, the managed system must increment $\mathbf{C_P}$ by one (1) and save its new value back into non-volatile storage.

2.26 Extended LAN Configuration Data

Table 2-21, *Extended LAN Configuration Data*, lists new configuration parameters for IPMI v2.0/RMCP+. Parameters are read-write and non-volatile unless otherwise specified. The IPMI *Set/Get LAN Configuration Parameters* commands are used for accessing this configuration data.

Table 2-21, *Extended LAN Configuration Data*

Parameter	#	Parameter Data (non-volatile unless otherwise noted) ¹
802.1q VLAN ID (12-bit)	20	Byte 1: [7:0] - Least significant 8-bits of the VLAN ID. 00h if VLAN ID not used. Byte 2: [7] - VLAN ID enable. 0b = disabled, 1b = enabled. If enabled, the BMC will only accept packets for this channel if they have 802.1q fields and their VLAN ID matches the [6:4] - reserved [3:0] - most significant four bits of the VLAN ID
802.1q VLAN Priority	21	Byte 1: [7:5] - reserved [2:0] - Value for Priority field of 802.1q fields. Ignored when VLAN ID enable is 0b (disabled) - See <i>802.1q VLAN ID</i> parameter, above. Setting is network dependent. By default, this should be set to 000b.
IPMI v2.0 Messaging Fixed Cipher Suite Entry Support (Read Only)	22	This parameter provides a count of the number (8 max.) of fixed Cipher Suites for use with IPMI Messaging on the given channel. Software can find out what security algorithms are associated with given Cipher Suite ID by using the <i>Get Channel Cipher Suites</i> command. In addition, there are Cipher Suite IDs assigned for standard Cipher Suites (see <i>Table 2-6, Cipher Suite IDs</i>) byte 1: [7:4] - reserved [3:0] - Fixed Entry count Number of configurable Cipher Suite entries, 1-based, 8h max. Configurable entries follow the non-configurable entries. E.g. if there are eight total entries, and three are configurable, then entries A:E would be non-configurable and F:H would be configurable.
IPMI v2.0 Messaging Fixed Cipher Suite Entries	23	This parameter contains zero to eight (8) bytes of fixed Cipher Suite IDs for Cipher Suites that can be used for establishing an IPMI messaging session with the BMC. The number of bytes supported is given in the preceding parameter. byte 1: Cipher Suite ID entry A. 00h = unspecified. byte 2: Cipher Suite ID entry B. 00h = unspecified. ... byte 8: Cipher Suite ID entry H. 00h = unspecified.

Parameter	#	Parameter Data (non-volatile unless otherwise noted) ¹
IPMI v2.0 Messaging Fixed Cipher Suite Privilege Levels	24	This parameter allows the configuration of which privilege levels are associated with each fixed Cipher Suite. The total number of nibbles supported (zero to eight) matches the number of fixed Cipher Suite IDs. byte 1: Maximum Privilege Level for 1st and 2nd Cipher Suites [7:4] - Maximum Privilege Level for 2nd Cipher Suite [3:0] - Maximum Privilege Level for 1st Cipher Suite 0h = Unspecified (given Cipher Suite is unused) 1h = Callback level 2h = User level 3h = Operator level 4h = Administrator level 5h = OEM Proprietary level byte 2: Maximum Privilege Level for 3rd and 4th Cipher Suites byte 3: Maximum Privilege Level for 5th and 6th Cipher Suites byte 4: Maximum Privilege Level for 7th and 8th Cipher Suites
IPMI v2.0 Messaging Configurable Cipher Suite Entry Support (Read Only)	25	This parameter provides a count of the number (8 max.) of configurable Cipher Suites for use with IPMI Messaging on the given channel. byte 1: [7:4] - reserved [3:0] - Configurable Entry Count. Total number of bytes of Configurable Cipher Suite entries in the following parameter. 1-based. 8h max.
IPMI v2.0 Messaging Configurable Cipher Suite Entries	26	This parameter contains zero to eight (8) bytes of configurable Cipher Suite IDs for Cipher Suites that can be used for establishing an IPMI messaging session with the BMC. Note: this parameter may be unsupported if the preceding parameter (Configurable Entry Count) = 0h. byte 1: Cipher Suite ID entry A. 00h = unspecified. byte 2: Cipher Suite ID entry B. 00h = unspecified. ... byte 8: Cipher Suite ID entry H. 00h = unspecified.
IPMI v2.0 Messaging Fixed Cipher Suite Privilege Levels	27	This parameter allows the configuration of which privilege levels are associated with each fixed Cipher Suite. The number of nibbles supported (zero to eight) matches the number of configurable Cipher Suite IDs. byte 1: Maximum Privilege Level for configurable Cipher Suites A & B [7:4] - Maximum Privilege Level for Cipher Suite B [3:0] - Maximum Privilege Level for Cipher Suite A 0h = Unspecified (given Cipher Suite is unused) 1h = Callback level 2h = User level 3h = Operator level 4h = Administrator level 5h = OEM Proprietary level byte 2: Maximum Privilege Level for configurable Cipher Suites C & D byte 3: Maximum Privilege Level for configurable Cipher Suites E & F byte 4: Maximum Privilege Level for configurable Cipher Suites G & H

3. RMCP+ Support and Payload Commands

The following sections list the commands associated with discovering, enabling, and activating payloads under IPMI v2.0/RMCP+. Also included in this section are updates and additions to IPMI commands to support IPMI v2.0/RMCP+ sessions, authentication, and configuration.

Table 3-1, RMCP+ Support and Payload Commands

Command	Section Defined	O/M
Activate Payload	3.1	O ^[1]
Deactivate Payload	3.2	O ^[1]
Suspend/Resume Payload Encryption Command	3.3	O ^[4]
Get Payload Activation Status	3.4	O ^[1]
Get Payload Instance Info	3.5	O ^[1]
Set User Payload Access Command	3.6	O ^[1]
Get User Payload Access Command	3.7	O ^[1]
Get Channel Payload Support Command	3.8	O ^[1]
Get Channel Payload Version Command	3.9	O ^[1]
Get Channel OEM Payload Info Command	3.10	O ^[1,2]
Set User Password	3.11	O ^[3]
Set Channel Security Keys	3.12	O

1. Mandatory if standard payloads other than IPMI Messaging are available on the channel.
2. Mandatory if OEM payloads are available on the channel.
3. Mandatory if session-based channels are supported.
4. Mandatory for standard payloads per *Table 3-4, Payload-specific Encryption Behavior*, when a channel supports CipherSuites that allow a session to be established with a standard Cipher Suite that supports encryption.

3.1 Activate Payload Command

This command is used for activating and deactivating a payload type under a given IPMI session. The ability to execute this command is determined via the user's privileges as assigned via the *Set User Payload Access* command.

The *Activate Payload* command may return a port number that is separate from the port number for the session that the command was issued under. In this case, the remote console must establish a session on the port number that the *Activate Session* command returned. The remote console must then issue the *Activate Payload* command on that port number in order to actually activate the payload. It is possible that the remote console already had a session active on the given port number. If the privileges associated with that session are sufficient (this will typically be the case unless the remote console activated the session at a privilege level that was lower than the maximum level for the user) the remote console can re-use the existing session and just use the *Activate Payload* command to activate the new payload type.

BMCs may have limited resources for handling multiple sessions. It is highly recommended that a remote console avoids creating multiple sessions and shares sessions for multiple payloads whenever possible.

Table 3-2, Activate Payload Command

	byte	data field
Request Data	1	[7:6] - reserved [5:0] - payload type (See Table 2-17, Payload Type Numbers)
	2	Payload Instance [7:4] - reserved [3:0] - payload instance. 1-based. 0h = reserved.
	3:6	Auxiliary Request Data. Additional payload-specific parameters to configure behavior of the payload when it becomes activated. Ignored if no auxiliary data is specified for given payload type. For Payload Type = SOL: <u>byte 1</u> [7] - Encryption Activation 1b: Activate payload with encryption. All SOL payload data from the BMC will be encrypted, if encryption was negotiated at the time of session activation. 0b: Activate payload without encryption. BMC will send all SOL payload data unencrypted, if that option is allowed. (An SOL configuration parameter allows a system to be configured to require encryption for all SOL transfers) [6] - Authentication Activation [5:4] - reserved [3:2] - Shared Serial Alert Behavior The following settings are determine what happens to serial alerts if IPMI over Serial and SOL are sharing the same baseboard serial controller. 11b: Reserved 10b: Serial/modem alerts succeed while SOL active. 01b: Serial/modem alerts deferred while SOL active. 00b: Serial/modem alerts fail while SOL active. [1] - Reserved [0] - SOL Activation Request The BMC asserts RTS and DCD/DSR to the baseboard serial controller and starts accepting characters as soon as SOL is activated. 1b: Request Activate SOL (no-op if SOL already activated) 0b: Deactivate SOL (no-op if SOL already deactivated) <u>byte 2:4</u> reserved - write a 00h
Response Data	1	Completion Code Generic plus the following command-specific completion codes: 80h: Payload already active on another session (required). This will be returned any time an attempt is made to activate a payload type when that type is already activated for another session, and when the BMC only supports one instance of that payload type running at a time. 81h: Payload type is disabled (optional). Given payload type is not configured to be enabled for activation. 82h: Payload activation limit reached. Cannot activate given payload type because the maximum number of simultaneous instances of that payload type are already running. 83h: Cannot activate payload with encryption. 84h: Cannot activate payload without encryption. BMC requires encryption for all payloads for given privilege level.
	2:5	Auxiliary Response Data. For Payload = SOL: reserved. Return as 00_00_00_00h.
	6:7	Inbound Payload Size Maximum size of payload data field from remote console to BMC. Excludes size of confidentiality header and trailer fields, if any. 1-based.
	8:9	Outbound Payload Size Maximum size of payload data field from BMC to remote console. Excludes size of confidentiality header and trailer fields, if any. 1-based.

10:11	Payload UDP Port Number UDP port number that payload can be transferred over. If the port number is same as the port that was used to establish the IPMI session, then SOL payload transfers are now available under that IPMI session on that port. Otherwise, the remote console will need to establish a separate IPMI Session to the specified Port Number using the same IP Address, username and password/key information that was used to establish the IPMI session. SOL Ppayload transfers will then be available over that session. If the remote console already has an IPMI session established on that port for a different payload type, the SOL payload type will now also be available over that session - provided that the session was established at a privilege level that matches the privilege level and authentication required for SOL. Otherwise, the remote console will need to close that session and re-establish it at the necessary privilege level.
12:13	Payload VLAN Number - FFFFh if VLAN addressing is not used.

3.2 Deactivate Payload Command

This command is used to terminate use of a given payload on an IPMI session. This type of traffic then becomes freed for activation by another session, or for possible re-activation under the present session. The *Deactivate Payload* command does not cause the session to be terminated. The *Close Session* command should be used for that purpose. A remote console application does not need to explicitly deactivate payload(s) prior to terminating a session. When a session terminates all payloads that were active under that session are automatically deactivated by the BMC.

Table 3-3, Deactivate Payload Command

	byte	data field
Request Data	1	[7:6] - reserved [5:0] - payload type (See Table 2-17, Payload Type Numbers)
	2	Payload Instance [7:4] - reserved [3:0] - payload instance. 1-based. 0h = reserved.
	3:6	Payload Auxiliary Data. Additional parameters to configure behavior of the payload when it becomes deactivated. Ignored if no auxiliary data is specified for given payload type. For Payload Type = SOL: (no auxiliary data) write as 0000 0000h
Response Data	1	Completion Code Generic plus the following command-specific completion codes: 80h: Payload already deactivated. 81h: Payload type is disabled (optional). Given payload type is not configured to be enabled for activation.

3.3 Suspend/Resume Payload Encryption Command

This command enables a remote console to control whether payload data from the BMC is sent encrypted or not. Since encryption can be a significant burden on software, this command provides a mechanism to allow higher performance by operating without encryption and only activating encryption when it is required for data confidentiality.

The extent at which this command can control encryption of data from the BMC is dependent on the payload definition. Some payload definitions may use a mix of encrypted and unencrypted payload data transfers. For example, a payload may implement a ‘request/response’ protocol, where the BMC would return an encrypted or unencrypted response based on whether the request from the remote console was encrypted or unencrypted. In this case, the command may only affect data that is autonomously generated by the BMC. Other payload definitions may just use whatever encryption the session was activated with, and offer no ‘run-time’ control of encryption/decryption, while other payload definitions may be ‘stream based’ where it is desirable for the remote console to be able to select when payload data is from the BMC is encrypted or not.

Table 3-4, Payload-specific Encryption Behavior

Payload Type = IPMI Messaging	
	- Encrypted requests from the remote console will get encrypted responses from the BMC. - The <i>Suspend/Resume Payload Encryption</i> command controls whether asynchronous (unrequested) messages from the BMC are encrypted or not. - PET Traps (which are actually separate from IPMI Messaging) are always sent unencrypted.
Payload Type = SOL	
	- The SOL configuration parameters allow configuring the system to require that SOL data be encrypted. - The BMC will transmit SOL payload data according to encryption settings that were selected when the payload was activated unless over-ridden by SOL configuration parameters. - The <i>Suspend/Resume Payload Encryption</i> command controls whether SOL Payload data is encrypted or not.

Table 3-5, Suspend/Resume Payload Encryption Command

	byte	data field
Request Data	1	[7:6] - reserved [5:0] - payload type (See <i>Table 2-17, Payload Type Numbers</i>)
	2	Payload Instance [7:4] - reserved [3:0] - payload instance. 1-based. 0h = reserved.
	3	[7:2] - reserved [4:0] - Operation 1h = Resume/Start encryption on all transfers of specified payload data from the BMC 0h = Suspend encryption on all transfers of specified payload messages from the BMC
Response Data	1	Completion Code Generic plus the following command-specific completion codes: 80h: Operation not supported for given payload type. 81h: Operation not allowed under present configuration for given payload type. 82h: Encryption is not available for session that payload type is active under. 83h: The payload instance is not presently active.

3.4 Get Payload Activation Status Command

This command returns what and how many instances of a given payload type are presently activated, and how many total instances can be activated.

Table 3-6, *Get Payload Activation Status Command*

	byte	data field
Request Data	1	Payload Type Number - number of the payload type to retrieve status for. See Table 2-17, Payload Type Numbers.
	2:4	OEM IANA (used to look up information by IANA and OEM Payload ID when Payload Type Number = "OEM Explicit", 00_00_00h otherwise.)
	5:6	OEM Payload ID (used to look up information by IANA and OEM Payload ID when Payload Type Number = "OEM Explicit" 0000h otherwise.)
Response Data	1	Completion Code
	2	Instance capacity [7:4] - reserved. [3:0] - Number of instances of given payload type that can be simultaneously activated on BMC. 1-based. 0h = reserved.
	3	[7] - 1b = instance 8 is activated. 0b = instance 8 is deactivated. [6] - 1b = instance 7 is activated. 0b = instance 7 is deactivated. ... [0] - 1b = instance 1 is activated. 0b = instance 1 is deactivated.
	4	[7] - 1b = instance 16 is activated. 0b = instance 16 is deactivated. [6] - 1b = instance 15 is activated. 0b = instance 15 is deactivated. ... [0] - 1b = instance 9 is activated. 0b = instance 9 is deactivated.

3.5 Get Payload Instance Info Command

This command returns information about a specific instance of a payload type. It is primarily used by software that may want to negotiate with an application that is presently using the given payload type. It accomplishes this by using the Session ID returned from this command with the *Get Session Info* command to look up the addressing information for the party that activated the payload. The application may then use that information to establish a direct dialog with the application that presently 'owns' the payload (note that this inter-application communication is not defined in the IPMI specifications).

Table 3-7, Get Payload Instance Info Command

	byte	data field
Request Data	1	Payload Type Number - number of the payload type to retrieve status for.
	2:4	OEM IANA (used to look up information by IANA and OEM Payload ID when Payload Type Number = "OEM Explicit", 00_00_00h otherwise.)
	5:6	OEM Payload ID (used to look up information by IANA and OEM Payload ID when Payload Type Number = "OEM Explicit" 0000h otherwise.)
	7	Payload Instance. 1-based. 0h = reserved.
Response Data	1	Completion Code
	2:5	Session ID - ID of session that instance is presently activated on. 00_00_00_00h if given instance is not activated. Remote software can use this information with the <i>Session Info</i> command to
	4:11	Payload-specific information (8-bytes) For Payload Type = SOL: <u>Byte 1</u> : Port Number A number representing the system serial port that is being redirected. 1-based. 0h = unspecified. Used when more than one port can be redirected on a system. <u>Byte 2:8</u> = reserved.

3.6 Set User Payload Access Command

IPMI v2.0 introduces the *Set User Payload Access* and *Get User Payload Access* commands. These commands can be thought of as extensions to the *Set User Access* and *Get User Access* commands, respectively. The *Set User Payload Access* command controls whether the specified user has the ability to activate the specified payload type on the given channel.

The *Set User Payload Access* command uses bitfields to allow a configuration utility to use a single command to set enable/disable multiple payloads at a time. Standard payloads are set separately from OEM payload enables. The command would be issued at least once with Standard payloads selected to set the configuration for Standard payloads, and then at least once with OEM Payloads selected to set the configuration for OEM payloads.

Table 3-8, Set User Payload Access Command

	byte	data field
Request Data	1	Channel Number [7:4] - reserved [3:0] - Channel Number
	2	User ID Standard/OEM Select [7] - 1b = set OEM Payload Enables 0b = set Standard Payload Enables [6] - reserved. User ID [5:0] - User ID. 000000b = reserved.
	3	Payload enables 1 <u>For Standard/OEM Select = 0b (standard):</u> [7:6] - reserved for standard payload enable 3 [5:4] - reserved for standard payload enable 2 [3:2] - standard payload enable 1 (SOL Payload enable) Controls whether the user can activate SOL Payloads. [1:0] - standard payload enable 0 (IPMI v2.0 messaging payload enable) Controls whether the user can use IPMI messages beyond those required for supporting any other enabled payloads. 00b = don't change present payload enable setting 01b = enable user to activate payload on this channel 10b = reserved 11b = disable user ability to activate payload on this channel <u>For Standard/OEM Select = 1b (OEM):</u> [7:6] - OEM 3 payload enable 00b = don't change present payload enable setting 01b = enable user to activate payload on this channel 10b = reserved 11b = disable user ability to activate payload on this channel [5:4] - OEM 2 payload enable [3:2] - OEM 1 payload enable [1:0] - OEM 0 payload enable
	4	Payload Enables 2 <u>For Standard/OEM Select = 0b (standard):</u> [7:6] - reserved for standard payload enable 7 [5:4] - reserved for standard payload enable 6 [3:2] - reserved for standard payload enable 5 [1:0] - reserved for standard payload enable 4 <u>For Standard/OEM Select = 1b (OEM):</u> [7:6] - OEM 7 payload enable [5:4] - OEM 6 payload enable [3:2] - OEM 5 payload enable [1:0] - OEM 4 payload enable
	5	Payload Enables 3 reserved - write as 00h.
	6	Payload Enables 4 reserved - write as 00h.
Response Data	1	Completion Code. Note: an implementation will not return an error completion code if the user access level is set higher than the privilege limit for a given channel. If it is desired to bring attention to this condition, it is up to software to check the channel privilege limits set using the <i>Set Channel Access</i> command and provide notification of any mismatch.

3.7 Get User Payload Access Command

The *Get User Payload Access* command returns the user payload enable settings that were set using the *Set User Payload Access* command.

Table 3-9, Get User Payload Access Command

	byte	data field
Request Data	1	Channel Number [7:4] - reserved [3:0] - Channel Number
	2	User ID Standard/OEM Select [7] - 1b = get OEM Payload Enables 0b = get Standard Payload Enables [6] - reserved. User ID [5:0] - User ID. 000000b = reserved.
Response Data	1	Completion Code
	2	Payload enables 1 <u>For Standard/OEM Select = 0b (standard):</u> [7:6] - reserved for standard payload enable 3 [5:4] - reserved for standard payload enable 2 [3:2] - standard payload enable 1 (SOL Payload enable) Controls whether the user can activate SOL Payloads. [1:0] - standard payload enable 0 (IPMI v2.0 messaging payload enable) Controls whether the user can use IPMI messages beyond those required for supporting any other enabled payloads. 00b = don't change present payload enable setting 01b = enable user to activate payload on this channel 10b = reserved 11b = disable user ability to activate payload on this channel <u>For Standard/OEM Select = 1b (OEM):</u> [7:6] - OEM 3 payload enable 00b = don't change present payload enable setting 01b = enable user to activate payload on this channel 10b = reserved 11b = disable user ability to activate payload on this channel [5:4] - OEM 2 payload enable [3:2] - OEM 1 payload enable [1:0] - OEM 0 payload enable
	3	Payload Enables 2 <u>For Standard/OEM Select = 0b (standard):</u> reserved - write as 00h. <u>For Standard/OEM Select = 1b (OEM):</u> [7:6] - OEM 7 payload enable [5:4] - OEM 6 payload enable [3:2] - OEM 5 payload enable [1:0] - OEM 4 payload enable
	4	Payload Enables 3 reserved.
	5	Payload Enables 4 reserved.

3.8 Get Channel Payload Support Command

This command enables local and remote console software to determine what payloads are enabled on the given BMC. The command returns a bitfield indicating which Payload Type numbers can be activated on the given channel.

Table 3-10, Get Channel Payload Support Command

	byte	data field
Request Data	1	Channel Number [7:4] - reserved [3:0] - Channel Number
Response Data	1	Completion Code
	2	[7] = Standard payload type #7 supported ... [0] = Standard payload type #0 supported
	3	[7] = Standard payload type #15 (0Fh) supported ... [0] = Standard payload type #8 supported
	4	[7] = Session Setup payload type #7 supported ... [0] = Session Setup payload type #0 supported
	5	[7] = Session Setup payload type #15 (0Fh) supported ... [0] = Session Setup payload type #8 supported
	6	[7] = Payload type 27h (OEM7) used ... [0] = Payload type 20h (OEM0) used
	7	[7] = Payload type 2Fh (OEM15) used ... [0] = Payload type 28h (OEM8) used
	8:9	reserved. Return as 0000h

3.9 Get Channel Payload Version Command

This command returns version information for the given payload type. The version number has major and minor parts. The major part of the version should only increment when there are significant changes to the payload format, commands, or payload-specific protocols that break backward compatibility with earlier versions. The minor part of the version increments when there are extensions to the payload format that are significant but are backwards compatible with earlier versions under the same major version number. An example of a major change would be a change to the payload activation process that would prevent earlier applications from activating the given payload type. An example of a minor format version change would be the definition of commands for new functions that did not exist under the previous format, but if unused, do not interfere with the operation of older applications.

Table 3-11, Get Channel Payload Version Command

Request Data	1	Channel Number [7:4] - reserved [3:0] - Channel Number
	2	Payload Type Number. (See Table 2-17, Payload Type Numbers)
	3:5	OEM IANA. When Payload Type Number is 02h (OEM Explicit) this field holds the OEM IANA for the OEM payload type to return version information for. Otherwise, this field is set to 00_00_00h.
	6:7	OEM Payload ID. When Payload Type Number is 02h (OEM Explicit) this field holds the OEM Payload ID for the OEM payload type to return version information for. Otherwise, this field is set to 0000h.
Response Data	1	Completion Code
	2	Format Version [7:4] - Major Format Version. BCD encoded (0 to 9) [3:0] - Minor Format Version. BCD encoded. (0 to 9) Software should present version data to the user in the format "major.minor" - e.g. 10h → "1.0"

3.10 Get Channel OEM Payload Info Command

This command provides a mechanism for software to determine the OEM Payload Type Number that corresponds to a particular type of OEM Payload, or vice versa. The command also returns the format version of the payload.

Table 3-12, Get Channel OEM Payload Info Command

Request Data	1	Channel Number [7:4] - reserved [3:0] - Channel Number
	2	Payload Type Number. (See Table 2-17, Payload Type Numbers). Use "OEM Explicit" to look up information by OEM IANA/OEM Payload ID.
	3:5	OEM IANA. When Payload Type Number is 02h (OEM Explicit) this field holds the OEM IANA for the OEM payload type to look up information for. Otherwise, this field is set to 00_00_00h.
	6:7	OEM Payload ID. When Payload Type Number is 02h (OEM Explicit) this field holds the OEM Payload ID for the OEM payload type to look up information for. Otherwise, this field is set to 0000h.
Response Data	1	Completion Code 80h = OEM Payload IANA and/or Payload ID not supported.
	2	Payload Type Number. (See Table 2-17, Payload Type Numbers) This is always returned as the OEM Payload Type number (OEM0-OEM7). "OEM Explicit" is <i>not</i> returned for this parameter.
	3:5	OEM IANA. IANA for the OEM that has defined the OEM payload type.
	6:7	OEM Payload ID. Payload ID value, specified by the OEM the defined the payload type.
	8	Format Version [7:4] - Major Format Version. BCD encoded (0 to 9) [3:0] - Minor Format Version. BCD encoded. (0 to 9) Software should present version data to the user in the format "major.minor" - e.g. 10h → "1.0"

3.11 Set User Password Command

This command is used to set and change user passwords and to enable and disable User IDs. If no password protection is desired for a given user, the password must be stored as an ASCII null-string. The management controller firmware will force the remaining fifteen bytes to 00h and store the password as sixteen bytes of 00h.

If this command is not supported, then the implication is that only User 1 with a fixed, null password is supported.

The password is stored as a 16-byte or 20-byte (for IPMI v2.0) 'octet string'. All values (0-255) are allowed for every byte. The management controller does not check the format or interpret values that are passed in with the Set User Password command.

Software is allowed to place additional restrictions on what passwords can be entered, in which case it is the responsibility of configuration software and console software to stay in synch with that definition. For example, remote console software could restrict passwords to the printable ASCII character set in order to simplify direct keyboard entry.

If this is done, any companion configuration utility should ensure that the user does not configure the managed system with non-printable passwords. Otherwise, it would be possible for the management controller to be configured with passwords that could not be entered via the remote console utility.

Table 3-13, Set User Password Command

	byte	data field
Request Data	1	User ID. For IPMI v2.0, the BMC shall support 20-byte passwords for all supported user IDs that have configurable passwords. The BMC shall maintain an internal tag that indicates whether the password was set as a 16-byte or as a 20-byte password. A 16-byte password can be used in algorithms that call for a 20-byte password. In this case, the 16-byte password is padded with 0's to 20-bytes. The 'test password' operation shall return the 'test failed' error completion code if an attempt is made to test a password that was stored as a 20-byte password as a 16-byte password (per 'password size' bit 7), and vice versa. [7] - password size 1b = set 20-byte user password/key. 0b = set 16-bytes user password/key (IPMI v1.5 backward compatible) [6] - reserved. [5:0] - User ID. 000000b = reserved. (User ID 1 is permanently associated with User 1, the null user name).
	2	[7:2] - reserved [1:0] - operation 00b = disable user 01b = enable user 10b = set password 11b = test password
	For password size = 16 bytes:	
	3:18	16-byte Password data. This is a required, fixed length field when used for the set- and test password operations. If the password is entered as an ASCII string, it must be null (00h) terminated and 00h padded if the string is shorter than 16 bytes. This field need not be present if the operation is 'disable user' or 'enable user'. If this field is present for those operations, the BMC will ignore the data.
	For password size = 20 bytes:	
	3:22	20-byte Password data. This is a required, fixed length field when used for the set- and test password operations. If the password is entered as an ASCII string, it must be null (00h) terminated and 00h padded if the string is shorter than 16 bytes. This field need not be present if the operation is 'disable user' or 'enable user'. If this field is present for those operations, the BMC will ignore the data.
Response Data	1	Completion Code. Generic, plus following command-specific completion codes: 80h = password test failed (mandatory)

3.12 Set Channel Security Keys Command

The *Set Channel Security Keys* command provides a standardized interface for initializing system unique keys that are used for the pseudo-random number generator key (K_R) and the key-generation key (K_G) used for RMCP+. This is mainly to provide a common interface for BMCs that are not pre-configured with these values.

The command includes a mechanism that allows specified keys to be 'locked'. Once locked, the key value cannot be read back or rewritten via standard IPMI commands. It is possible, however, that a firmware update or re-installation procedure may cause the keys

to be cleared or unlocked. Software utilities responsible for BMC initial installation and setup should check to see whether keys have been locked and if not, should initialize them appropriately and lock them.

If this command is not supported, it indicates that the keys are either permanently pre-configured, or that they are only configurable via an OEM/BMC-specific mechanism.

Table 3-14, Set Channel Security Keys Command

	byte	data field
Request Data	1	Channel Number [7:4] - reserved [3:0] - Channel Number
	2	Operation [7:2] - reserved [1:0] - Operation 00b = read key BMC returns value of specified key, provided key has not yet been locked. Some BMCs may allow the key to be re-written if it does not match the expected value. Other BMCs may only allow one 'set' operation. If the key value has not yet been initialized, the BMC will return 0's for the key value. Utility software responsible for BMC installation and initial setup can use this Operation to also check to see whether keys have been initialized and locked. 01b = set key BMC writes given key value to non-volatile storage. 10b = lock key BMC locks out modification or reading the key value. Once a key has been locked, it is not cannot be rewritten or read via IPMI specified commands. 11b = reserved
	3	Key ID [7:0] - key ID. 00h = RMCP+ "KR" key 01h = RMCP+ "KG" key all other = reserved
	(4:M)	Key value. Value for specified key. Used for "set" Operation only. Otherwise, this field is not used in the request. The BMC will ignore any bytes following the 'Key ID' byte.
Response Data	1	Completion Code. Generic, plus following command-specific completion codes: 80h = Cannot perform set / confirm. Key is locked (mandatory) 81h = insufficient key bytes 82h = too many key bytes 83h = key value does not meet criteria for specified type of key
	(2:N)	Key value. The BMC returns data the specified key value when the Operation is set to "confirm key". Otherwise, the BMC returns no additional bytes past the completion code.

4. Serial Over LAN

Serial over LAN (SOL) is the specification of packet formats and protocols for transmitting serial data over LAN using IPMI over LAN packets. The typical goal of this

capability is to redirect the traffic to/from a local asynchronous serial controller interface. This enables communication over LAN with local software that only understands how to communicate through a local serial controller. This can be used for implementing a virtual remote serial terminal for enabling the user or remote software interaction with serial-based interfaces for operating systems (e.g. “command-line” interfaces and Linux console) and management services (e.g. Microsoft’s serial-based Emergency Management Services (EMS)).

4.1 SOL and Serial Port Sharing

Serial-over-LAN has been designed to be able to work alongside the Serial Port Sharing capability of IPMI. This supports an implementation where traffic to/from the serial controller interface on the baseboard can be routed either to the BMC, SOL, or to the system’s serial port connector. To accomplish this, the specifications define commands to control the routing of the serial stream using the serial multiplexer logic, also referred to as the ‘mux’. *Table 4-1, Mux Settings* describes the basic connections that support Serial-over-LAN. When Serial Port Sharing is used with SOL, the mux has three settings:

Table 4-1, Mux Settings

Setting	Description
System	The mux connects the serial connector on the chassis to the system serial controller. The RxD line is routed so the BMC can snoop incoming serial traffic for escape sequences and character patterns.
BMC	The mux connects the serial connector on the chassis to the BMC. The mux circuitry also places the hardware handshake lines to the baseboard serial controller into a steady state.
SOL	The mux is set to connect the baseboard serial controller directly to the BMC.

Note that there is no requirement that SOL is used with serial port sharing. SOL can be implemented with a dedicated serial controller interface where the interface only provides traffic for SOL connections. Serial Port Sharing can be implemented on a separate port if desired, or not at all.

The IPMI specifications do not specify the hardware design and implementation of SOL or Serial Port Sharing. *Figure 4-1, SOL with Serial Port Sharing*, presents an example block diagram for the purposes of illustrating the concept of SOL when used with Serial Port Sharing.

The example figure shows the signal routing when the mux is set to ‘SOL’. The bold lines represent the flow of data. The interconnections and blocks shown are to illustrate the functional relationships between the system management elements, and do not map directly to the exact circuit implementation of the architecture.

When the mux is set to ‘SOL’, the serial connection at the back of the box is isolated from both the baseboard serial controller and the BMC and the serial connector cannot be used for communicating with the BMC or the system. However, when SOL is not in use, the BMC can allow the serial connector to be used for functions such as the communicating with the BMC via IPMI over Serial, or as a regular serial port.

Refer to Section 4.10, *SOL Interaction with Windows.NET Escape Sequences*, and Section 4.11, *SOL Payload Activated with Serial Port Sharing* for additional information on the interaction between SOL and Serial Port Sharing.

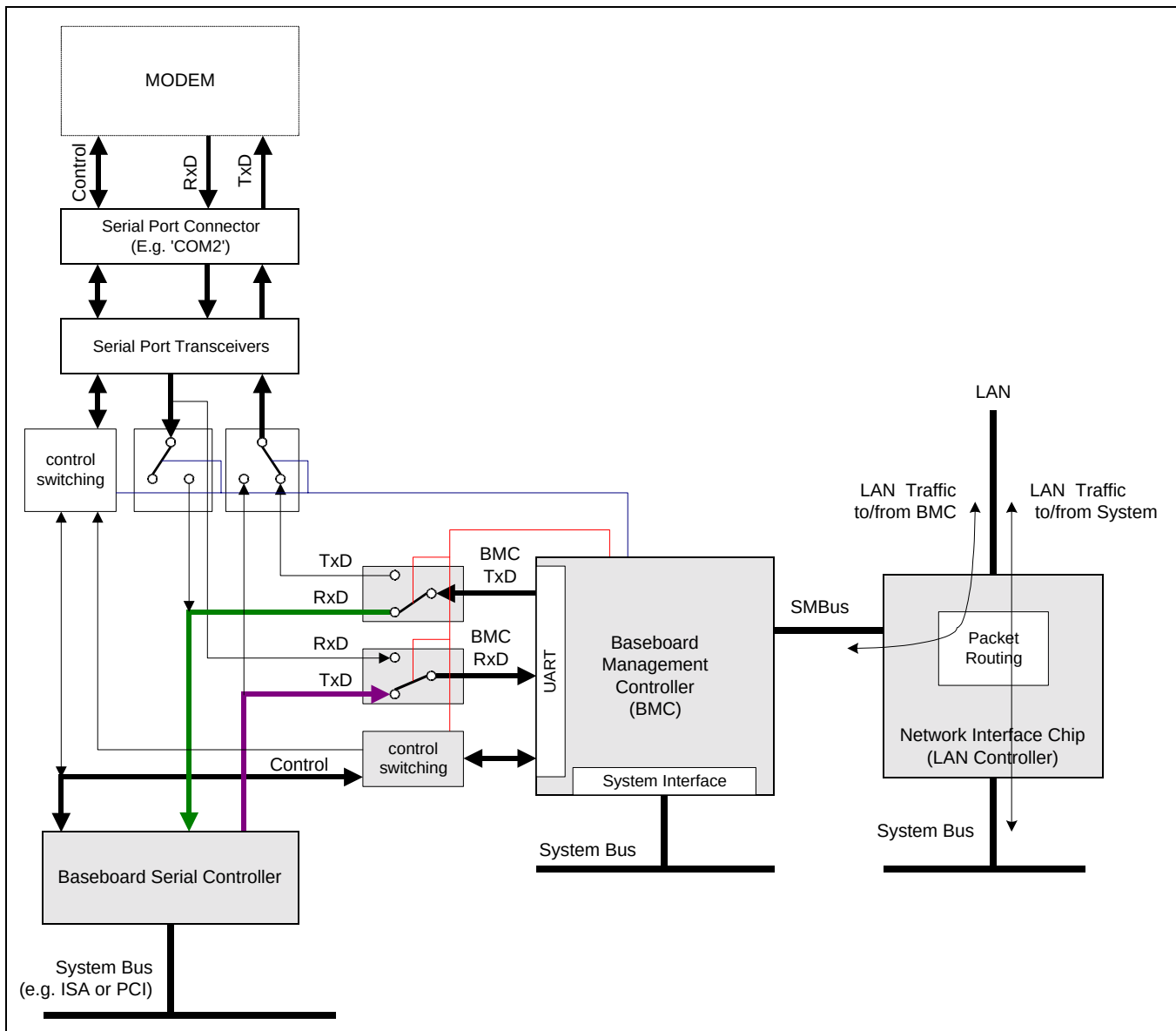


Figure 4-1, SOL with Serial Port Sharing

4.2 SOL Operation Overview

SOL operation is conceptually straightforward. A remote management application can establish an IPMI-over-LAN session with the BMC. Once the session is established, the remote console can request that the SOL be activated. If SOL is used with Serial Port Sharing, this causes the BMC to set the mux to 'SOL'.

From this point, any outgoing characters from the baseboard serial controller are assembled into packets by the BMC and sent to the remote console over the LAN. Conversely, in-bound LAN packets carrying characters for the system serial controller have their character data extracted by the BMC and delivered to the baseboard serial controller.

The SOL character data is carried as SOL Messages that are carried in UDP datagrams. The packet format is that for IPMI v2.0 RMCP+ with the Payload Type set to “SOL”. The SOL Payload includes fields that are used for supporting acknowledge and retries of SOL messages, and for supporting functions such as flushing buffers or temporarily suspending serial traffic using flow control.

4.3 SOL Security

Authentication and encryption are handled at the RMCP+ level.

4.4 SOL Sequence Numbers

At the session level, SOL Payloads share the session sequence numbers for authenticated and unauthenticated packets with other packets under the IPMI session. At the payload level, SOL packets include their own message sequence numbers that are used for tracking missing and retried SOL messages.

4.5 Flow Control

Flow Control is used to help ensure that serial data is not lost because of differences in the throughput between the serial controller interface and the network interface. When SOL is used with a physical serial controller, flow control on the serial controller side is accomplished by the BMC controlling and monitoring the hardware handshake signal lines (RTS, CTS, DCD/DSR, DTR) on the serial controller.

Some implementations may have the serial controller function integrated into the BMC or another device. Such implementations may not have physical hardware flow control lines, but there must be internal control capabilities that accomplish the same thing.

Flow control on the network side is handled by use of acknowledges (ACKs) and negative-acknowledges (NACKs) that indicate whether the BMC is ready to accept more data or not.

4.6 Bit Rate Handling

Some implementations of SOL will connect pins for a serial interface on the BMC to the pins for the serial interface of a serial controller for the system. For standard microcontroller serial interfaces, the BMC would need to know the bit rate setting of the motherboard serial controller in order for the BMC’s serial controller to be in synch. Thus, there is a non-volatile configuration setting for setting the bit rate for SOL in the BMC.

Other implementations may incorporate an embedded serial controller in the BMC or may have hardware mechanisms that allow the BMC to get the bit rate setting directly from the system serial controller. In this case, the non-volatile configuration bit rate setting is not used.

4.7 SOL Payload Data Format

Table 4-2, *SOL Payload Data Format*, specifies the fields that make up the SOL Payload in an RMCP+ packet.

Table 4-2, *SOL Payload Data Format*

Field	Size	Description
Packet Sequence Number	1	Sequence Number for this packet. Sequence numbers must be non-zero. Multiple outstanding sequence numbers are not supported in this version of the specification. Retried packets use the same sequence number as the first packet. [7:4] Reserved [3:0] Packet sequence number. 0h = ACK-Only packet.
Packet ACK /NACK Sequence Number	1	Sequence Number for packet being ACK'd or NACK'd. [7:4] Reserved. Write as 0h. (Future spec may use this to specify a range of packets being acknowledged) [3:0]: Packet sequence number being ACK'd/NACK'd. 0h: Informational packet. No request packet being ACK'd or NACK'd.
Accepted Character count	1	<u>Accepted Character Count</u> 1-based. This field indicates the number of characters accepted from the packet, if any. 00h = Packet received but no data bytes accepted. For BMC-to-Console: In order to improve throughput, the BMC is allowed to append additional characters to a packet when it resends it. To support this, the remote console must accept retried packets (packets with the same packet sequence number) and check to see if the packet contains additional data. If the packet does contain addition data, the remote console should accept the data and acknowledge the packet using the packet sequence number and return the count of the number of characters that it had received. <i>The console must either accept all packet data sent to it or NACK the entire packet. It is not allowed to accept partial packet data.</i> For Console-to-BMC: The BMC is allowed to accept partial packet data by NACK'ing the packet and returning a character offset that is less than the data length sent by the remote console. The remote console would then send the next packet starting with the first byte of data that the BMC rejected.
Operation / Status	1	BMC to Remote Console:
		Remote Console to BMC:
		Operations are executed <i>before</i> character data is transferred. [7] reserved [6]
		Note: Operations are executed <i>before</i> character data is transferred. [7] reserved [6] ACK/NACK

Field	Size	Description
		1b: Packet is being NACK'd. The BMC is unable to accept all character data from packet. Note: Operation field is still accepted even if packet is NACK'd. 0b: ACK. BMC ready to accept next packet of character data. [5]^[1] A NACK packet with this status will be automatically sent one time after this bit changes state. (Whenever the system enters or leaves a power state where character transfers to the system serial controller are possible) A NACK packet with "Character transfer is unavailable" status will also be sent for each character transfer request from the remote console when the system is in a powered-down or sleep state. 1b: Character transfer is unavailable because system is in a powered-down or sleep state. 0b: SOL character transfer is available. [4]^[2] A NACK packet with this status will be automatically sent one sent once, just before the BMC deactivates SOL because of a front panel power-button or a reset. 1b: SOL is deactivated/deactivating. [Remote console can use this to tell if SOL was deactivated by some other party, or by local pushbutton reset or power on/off]. 0b: SOL is active. [3] Transmit Overrun 1b: characters were dropped between transmitting this packet and the previous packet, because the system did not pay attention to hardware flow control. 0b: no characters were lost between this packet and the preceding packet. [2] Break 1b: A break condition from the system has been detected. The BMC will generate this 1b: NACK. Packet is being NACK'd by the remote console. 0b: ACK. Packet is being ACK'd by the remote console. [5] Ring/WOR Assert RI (may not be supported on all implementations) - Goal is to allow this to be used for generating a WOR. [4] Break 1b: Generate BREAK (300 ms, nominal) [3] CTS Pause 1b: Deassert CTS (clear to send) to the baseboard serial controller. This is the default state when SOL is deactivated. 0b: Let BMC control CTS. [2] Drop DCD/DSR 1b: Deassert DCD/DSR to baseboard serial controller 0b: Assert DCD/DSR to baseboard serial controller. [1] Flush Inbound 1b: Drop (flush) data from remote console to BMC [not including data carried in this packet, if any] [0] Flush Outbound 1b: Flush Outbound Character Data (flush data from BMC to remote console)

Field	Size	Description
		only on one packet at the start of the break. 0b: no break detected [1:0] Reserved
Character Data	varies	Data length, in bytes, is equal to the IPMI Message/SOL Message Length field in the Session Header, minus the bytes for the Packet Sequence Number, through and including the Operation Field.

1. If the system is powered down or in certain sleep states, the baseboard serial controller will not be available for transferring characters. When the BMC receives data from the remote console, and the system is powered down or sleeping, the BMC can use this bit to return status to indicate to the remote console why the characters it received may not be able to be transferred to the system. Note that this is an 'advisory' bit. The BMC will still attempt to put characters into the system serial controller in case that the serial controller is configured to wake the system under this condition. It is mandatory that the BMC returns this bit when the system is powered down (S4/S5). The BMC may not be able to differentiate other system sleep states, in which case the bit should be returned as 0b (transfers available).
2. The BMC issues this status when the payload is deactivated for due to the following conditions: a. The payload is deactivated because of a manual power down or reset. (An implementation is recommended to have a local manual "pushbutton" reset or power-off to deactivate an SOL payload. This is provided as a way of terminating remote control connections for local servicing.) b. The SOL payload is deactivated via the *Deactivate Payload* or *Close Session* commands. These command may have been issued from another session. (System software operating through the system interface, and users with Admin privilege have the ability to issue the *Deactivate Payload* and *Close Session* commands to other sessions).

4.8 Activating SOL using RMCP+ Authentication

To use SOL a remote console or remote application must first establish an IPMI Session with the BMC. This is accomplished by sending the specified IPMI and RMCP+ RSSP/RAKP messages to the BMC with the appropriate user name, role, and password/key information. If the remote console plans to use encryption with SOL, the console must also negotiate an encryption algorithm at the time that the IPMI session is established.

Once the IPMI session has been established, the *Get Channel Payload Support* command can be used to retrieve the present availability of SOL and the version of SOL.

If SOL is not already active on another session, the next step is to issue the *Activate Payload* command. The command will return information about what serial input and output data buffer sizes are available on the BMC as well as the UDP port number over which SOL packets can be transferred.

The port that was used for establishing the IPMI session may not be the same port number that SOL is available over. Some implementations transfer SOL payloads available over a separate UDP port in order to provide better performance. If the *Activate Payload* command returns a port number that is different than the port number that was used to establish the IPMI session, the remote console must establish a separate IPMI Session to the specified port number using the same IP Address, username and password/key information that was used to establish the IPMI session.

Note that if a second session has already been established on that port number for a different payload type, the existing session can also be used for SOL payloads, provided

that the session was established at a privilege level that matches the privilege level and authentication required for SOL. Otherwise, the remote console will need to close that session and re-establish it at the necessary privilege level.

4.9 SOL Packet Acknowledge and Retries

A packet acknowledge is of one of two types:

- An ACK, indicating that the packet has been received and all its data has been accepted
- A NACK, indicating that the packet was received but some or all of the data could not be accepted

To improve efficiency, the packet acknowledgment information can be carried in a packet that also carries the SOL character data. Conversely, a packet can be an ACK-only packet that carries ACK or NACK information, but no data. However, packets are only acknowledged if they contain character data. Therefore, ACK-only packets themselves are not acknowledged.

Except for ACK-only packets, the remote console must acknowledge each SOL packet that it receives. If the BMC does not receive an ACK packet within a timeout interval, the BMC will resend (retry) the packet. The number of retries and the amount of time between retries are configurable through the SOL Configuration Parameters. **Once the number of retries has been met the BMC will drop the packet and the data will be lost.**

If the remote console wants to temporarily stop the BMC from accepting characters from the system, it should use the “CTS Pause” bit in the control/status byte. Whether this stops the system from transmitting is dependent on whether the system software pays attention to the CTS (hardware handshake) status or not. If the system continues to send characters to the BMC, the BMC will attempt to transmit them to the remote console.

It is possible that additional characters could be received from the system serial controller while the BMC was waiting for a retry timeout. In order to improve throughput, the BMC is allowed the option of appending additional characters to a packet whenever it resends it. To support this, the remote console must accept retried packets (packets with the same packet sequence number) and check to see if the packet contains additional data. If the packet does contain additional data, the remote console should accept the data and acknowledge the packet using the packet sequence number and character offset value that it had received.

Table 4-3, Remote Console to BMC SOL Packet Handling

Packet fields from remote console:			Function / BMC Action
ACK/ NACK	ACK'd/NACK'd Seq#	Accepted Character Count	
ACK	Seq # from BMC data packet	Non-zero matches character count from BMC	“Completion ACK” BMC processing for specified packet is done. A packet from the remote console with an ACK and an Accepted Character Count for the full amount of data for the packet indicates that the remote console has successfully accepted the packet.
ACK	Seq # from BMC data packet	Non-zero less than character count from BMC	“Partial ACK” BMC immediately retransmits specified packet It is possible that the remote console may have missed a BMC retry where the BMC had appended more data to the packet (retry intervals should be configured to avoid this scenario). If the BMC receives an ACK for less than the last amount of transmitted data, the BMC will cease appending data to the packet and will retransmit the packet until it receives an ACK from the remote console with an Accepted Character Count for the full amount of packet data.
NACK	Seq # from BMC data packet	0	“Suspend NACK” BMC stops sending specified packet. The remote console would use the Suspend NACK if it were temporarily out of buffer space for characters already queued up in the BMC and did not want those characters to get dropped. The BMC stops sending / retrying specified packet and waits until it gets either a “Partial ACK”, “Completion ACK”, “Resume ACK” or a Flush Outbound operation from the remote console. The BMC will deassert CTS when it gets near running out of buffer space for characters from the system. If characters continue to come in (CTS is ignored by the system) a transmit overrun condition can occur.
ACK	Seq # from BMC data packet	0	“Resume ACK” BMC immediately retransmits specified packet. Cancels a “Suspend NACK”. A packet from the remote console with an ACK and an Accepted Character Count of zero (0) bytes will cause the BMC to immediately retransmit the packet with the corresponding sequence number to the remote console. This can be used as a way to get the BMC to restart transmission after a Suspend NACK from the remote console.
ACK	0h	0	“Control-only Packet” BMC performs operation specified in the control/status field. For FLUSH operation: A packet from the remote console with a “Flush Outbound” operation will cause the BMC to cease any retries in progress and the BMC will start accumulating character data anew. The remote console can use this as a recovery mechanism if it gets ‘lost’ in the sequence from the BMC. See packet format table for info on other functions.

4.10 SOL Interaction with Windows.NET Escape Sequences

The Microsoft .NET Emergency Management Services specification (See [MSFT EMS]) defines certain character sequences for performing the following operations:

System Hard Reset	<ESC>R<ESC>r<ESC>R
Invoke Service Processor (i.e. switch to BMC)	<ESC>(
Exit Service Processor (with optional prompt)	Q
Exit Service Processor (without confirmation prompt)	<ESC>Q

The specification also requires that a switch to the Service Processor is acknowledged by sending an <ESC>* sequence to the remote console:

Acknowledge Switch to Service Processor	<ESC>*
---	--------

Typically, the input escape sequences would be received by the BMC over the serial/modem connection. In this case, the sequences for Invoking and Exiting the service processor would control the serial/modem mux setting associated with serial port sharing. However, there are separate streams for SOL and BMC traffic, so unlike serial port sharing there is no need for a mux to switch between console traffic to the system serial controller and traffic to the BMC.

Therefore, since you can always send commands to the BMC the BMC itself does not snoop for or handle Windows .NET Escape sequences in the SOL character data. However, a remote console application can emulate support for the Windows .NET headless sequences by filtering for the Windows .NET escape sequences prior to placing the data in an SOL packet. If the Windows .NET escape sequences are detected, the remote console can then take the appropriate actions. For example, if the Reset sequence is detected, the remote console would send the IPMI command for a system reset.

4.11 SOL Payload Activated with Serial Port Sharing

The following lists the behavior of certain IPMI commands while SOL is activated. This applies only when the SOL is being used in conjunction with Serial Port Sharing.

- The *Set Channel Access* command is accepted for the serial channel, but while SOL is activated the Channel Access Mode is forced to 'disabled'. Attempting to change the access mode while SOL is activated will result in a D5h completion code. The access mode shall be saved when SOL is activated, and restored when SOL is deactivated.
- The *Set Serial/Modem Configuration Parameters* and *Get Serial/Modem Configuration Parameters* commands are accepted for the serial/modem channel and have no special behavior while SOL is activated.
- The UDP proxy commands, *Set PPP UDP Proxy Transmit Data*, *Get PPP UDP Proxy Transmit Data*, *Send PPP UDP Proxy Packet*, *Get PPP UDP Proxy Receive Data*, will receive a D5h error completion code while SOL is activated.

- The *Callback* command will receive a D5h error completion code when the command is targeted to the serial/modem channel being used for SOL and SOL is activated. The *Set User Callback Options* and *Get User Callback Options* commands will be accepted.
- The *Send Message Command* will receive a D5h error completion code when the command is targeted to the serial/modem channel being used for SOL and SOL is activated.
- The *Alert Immediate Command* will receive a D5h error completion code when the command is targeted to the serial/modem channel being used for SOL and SOL is activated.
- While SOL is activated, the *Set Serial/Modem Mux* command will respond to the requested operations as follows:

Table 4-4, Set Serial/Modem Mux Command Operation while SOL Active

Operation	Response
0h = get present mux setting/status only	Accepted. BMC returns that mux is set to 'System'
1h = request switch of mux to system	Accepted
2h = request switch of mux to BMC	Rejected (see response data for command)
3h = force switch of mux to system	Allowed
4h = force switch of mux to BMC	BMC returns D5h completion code
5h = block requests to switch mux to system	BMC returns D5h completion code
6h = allow requests to switch mux to system	Accepted
7h = block requests to switch mux to BMC	BMC returns D5h completion code
8h = allow requests to switch mux to BMC	Accepted

5. SOL Commands

The following commands are specific to the SOL Payload type.

Table 5-1, SOL Commands

Command	Section Defined	O/M
SOL Activating	5.1	O ^[1]
Set SOL Configuration Parameters	5.2	O ^[1]
Get SOL Configuration Parameters	5.3	O ^[1]

1. Mandatory if the SOL Payload type is implemented.

5.1 SOL Activating Command

This command provides a mechanism for the BMC to notify a remote application that a SOL payload is activating on another channel. The primary use for this command is in conjunction with Serial Port Sharing to notify an application that is connected to the system serial port that they have lost their connection because another application has activated SOL and the baseboard serial controller is now being used for SOL.

Table 5-2, SOL Activating Command

	byte	data field
Request Data	1	SOL Session State [7:4] - reserved [3:0] - Session State 0h = SOL Activating (Shared serial connection about to be lost because BMC is about to switch it over for SOL use)
	2	SOL Payload Instance
	3	SOL format version, major (SOL Payload Format version. See <i>Table 2-17, Payload Type Numbers</i>).
	4	SOL format version, minor (SOL Payload Format version. See <i>Table 2-17, Payload Type Numbers</i>).
Response Data	1	Completion Code The request message is a message that is asynchronously generated by the BMC. The BMC will not wait for a response from the remote console before dropping the serial connection to proceed with SOL, therefore the remote console does not need to respond to this command.

5.2 Set SOL Configuration Parameters Command

This command is used for setting parameters such as the network addressing information required for SOL payload operation.

Table 5-3, Set SOL Configuration Parameters Command

	byte	data field
Request Data	1	[7:4] - reserved [3:0] - Channel number.
	2	Parameter selector
	3:N	Configuration parameter data, per <i>Table 5-5, SOL Configuration Parameters</i>
Response Data	1	Completion Code 80h = parameter not supported. 81h = attempt to set the 'set in progress' value (in parameter #0) when not in the 'set complete' state. (This completion code provides a way to recognize that another party has already 'claimed' the parameters) 82h = attempt to write read-only parameter

5.3 Get SOL Configuration Parameters Command

This command is used for retrieving the configuration parameters from the *Set SOL Configuration* command.

Table 5-4, Get SOL Configuration Parameters Command

	byte	data field
Request Data	1	[7] - 0b = get parameter 1b = get parameter revision only. [6:4] - reserved [3:0] - Channel number.
	2	Parameter selector
	3	Set Selector. Selects a given set of parameters under a given Parameter selector value. 00h if parameter doesn't use a Set Selector.
	4	Block Selector (00h if parameter does not require a block number)
Response Data	1	Completion Code. Generic codes, plus following command-specific completion code(s): 80h = parameter not supported.
	2	[7:0] - Parameter revision. Format: MSN = present revision. LSN = oldest revision parameter is backward compatible with. 11h for parameters in this specification.
		<i>The following data bytes are not returned when the 'get parameter revision only' bit is 1b.</i>
	3:N	Configuration parameter data, per <i>Table 5-5, SOL Configuration Parameters</i> If the rollback feature is implemented, the BMC makes a copy of the existing parameters when the 'set in progress' state becomes asserted (See the Set In Progress parameter #0). While the 'set in progress' state is active, the BMC will return data from this copy of the parameters, plus any uncommitted changes that were made to the data. Otherwise, the BMC returns parameter data from non-volatile storage.

Table 5-5, SOL Configuration Parameters

Parameter	#	Parameter Data (non-volatile unless otherwise noted) ⁽¹⁾
Set In Progress (volatile)	0	<u>data 1</u> - This parameter is used to indicate when any of the following parameters are being updated, and when the updates are completed. The bit is primarily provided to alert software than some other software or utility is in the process of making changes to the data. An implementation can also elect to provide a 'rollback' feature that uses this information to decide whether to 'roll back' to the previous configuration information, or to accept the configuration change. If used, the roll back shall restore all parameters to their previous state. Otherwise, the change shall take effect when the write occurs. [7:2] - reserved [1:0] - 00b = set complete. If a system reset or transition to powered down state occurs while 'set in progress' is active, the BMC will go to the 'set complete' state. If rollback is implemented, going directly to 'set complete' without first doing a 'commit write' will cause any pending write data to be discarded. 01b = set in progress. This flag indicates that some utility or other software is presently doing writes to parameter data. It is a notification flag only, it is not a resource lock. The BMC does not provide any interlock mechanism that would prevent other software from writing parameter data while. 10b = commit write (optional). This is only used if a rollback is implemented. The BMC will save the data that has been written since the last time the 'set in progress' and then go to the 'set in progress' state. An error completion code will be returned if this option is not supported. 11b = reserved
SOL Enable	1	Byte 1: [7:1]: Reserved [0]: SOL Enable. Note, this controls whether the SOL payload type can be activated. Noter that whether an SOL stream can be established is also dependent on the Access Mode and Authentication settings for the corresponding LAN channel. The enabled/disabled state and access mode settings for the serial/modem channel have no effect on SOL. 1b = enable SOL payload 0b = disable SOL payload

Parameter	#	Parameter Data (non-volatile unless otherwise noted) ^[1]
SOL Authentication	2	Byte 1: SOL Authentication Enable [7]: Force SOL Payload Encryption 1b: Force encryption. If the Cipher Suite for the session supports encryption this setting will force the use of encryption for all SOL payload data. 0b: Encryption controlled by remote console. Whether SOL Packets are encrypted or not is selectable by the remote console at the time the payload is activated (using the <i>Activate Payload</i> command) and can be changed during operation via the <i>Suspend/Resume Payload Encryption</i> command. [6]: Force SOL Payload Authentication 1b: Force Authentication. If the Cipher Suite for the session supports authentication this setting will force the use of authentication on all SOL Payload data. 0b: Authentication controlled by remote software. Note that for the standard Cipher Suites, if encryption is used authentication must also be used. Therefore, while encryption is being used software will not be able to select using unauthenticated payloads. :4]: Reserved [3:0]: SOL Privilege Level. Sets the minimum operating privilege level that is required to be able to activate SOL using the <i>Activate Payload</i> command. 0h: reserved 1h: reserved 2h: USER level 3h: OPERATOR level 4h: ADMINISTRATOR level 5h: OEM Proprietary level all other: Reserved
Character Accumulate Interval & Character Send Threshold	3	Byte 1: Character Accumulate Interval in 5 ms increments. 1-based. This sets the typical amount of time that the BMC will wait before transmitting a partial SOL character data packet. (Where a partial packet is defined as a packet that has fewer characters to transmit than the number of characters specified by the Send Threshold - see next field in this parameter). A packet will not be sent 00h = reserved Byte 2: Character Send Threshold. 1-based. The BMC will automatically send an SOL character data packet containing this number of characters as soon as this number of characters (or greater) has been accepted from the baseboard serial controller into the BMC. This provides a mechanism to tune the buffer to reduce latency to when the first characters are received after an idle interval. In the degenerate case, setting this value to a '1' would cause the BMC to send a packet as soon as the first character was received. This can be useful if the character accumulate interval is large. If the BMC is waiting for an acknowledge from the previous packet, it will ignore this threshold and continue to collect data until it has a full packet's worth.

Parameter	#	Parameter Data (non-volatile unless otherwise noted) ^[1]
SOL Retry	4	Byte 1: Retry Count [7:3] - Reserved [2:0] - Retry count. 1-based. 0 = no retries after packet is transmitted. Packet will be dropped if no ACK/NACK received by time retries expire. Byte 2: Retry Interval. 1-based. Retry Interval in 10 ms increments. Sets the time that the BMC will wait before the first retry and the time between retries when sending SOL packets to the remote console. 00h: Retries sent back-to-back
SOL non-volatile bit rate (non-volatile)	5	This configuration parameter is not supported if the implementation does not have a BMC serial controller that can be potentially configured Serial communication with the BMC when SOL is activated always occurs using 8 bits/character, no parity, 1 stop bit, and RTS/CTS (hardware) flow control. Note: If SOL is enabled for multiple LAN channels, the BMC uses the serial communication settings for the channel over which the Activate SOL command was initially received. The settings for other channels are ignored. Data 1 [7:4] - Reserved [3:0] - Bit Rate. 1-5h = reserved. Support for bit rates other than 19.2 kbps is optional. The BMC must return an error completion if a requested bit rate is not supported. It is recommended that the parameter out-of-range (C9h) code be used for this situation. 0h: Use setting presently configured for IPMI over serial channel used by BIOS serial redirection. The setting will be used even if the access mode for the serial channel is set to 'disabled'. Note: IPMI specification can allow more than one serial channel. 6h: 9600 bps 7h: 19.2 kbps 8h: 38.4 kbps 9h: 57.6 kbps Ah: 115.2 kbps all other = reserved
SOL volatile bit rate (volatile)	6	Set volatile version of SOL Serial Settings. Data follows that for the SOL non-volatile bit rate parameter.

Appendix A - Previous Sequence Number Tracking

The following illustrates how a method for tracking the last eight received sequence numbers can be used for handling out-of-sequence packet reception. The method illustration assumes that the receiver tracks the highest received sequence number that has been accepted, shown as 'Highest Received' column, and which of the previous eight sequence numbers have been received, shown as the 'Previously Received List'.

Note that for implementation, the previously received list can be implemented as a bit field where 1=received, 0=not received, and the bit positions correspond to sequence numbers for Highest Received-1, Highest Received-2, Highest Received-3, etc. The handling of wrap-around of the sequence number is not shown and is left to the reader.

1. Startup initialization. Highest Received session sequence number is set to value that was sent with the *Activate Session* command. The Previously Received List is initialized as if the preceding eight sequence numbers were received. This prevents a packet with a sequence number less than the initial value from being accepted. For this example, assume the initial sequence number is 40.

Initial values are:

Highest Received	Previously Received List							
40	39	38	37	36	35	34	33	32
	Y	Y	Y	Y	Y	Y	Y	Y

2. Packet 41 is received. Packet is accepted because it is no more than 8 counts greater than the Highest Received value from step 1. Highest Received becomes 41. Previously received list gets 'pushed up'. Updated values are:

Highest Received	Previously Received List							
41	40	39	38	37	36	35	34	33
	Y	Y	Y	Y	Y	Y	Y	Y

3. Packet 44 received. Accepted because it is within 8 counts of the last Highest Received. Highest Received becomes 44. Previous received list gets pushed up by 2. Note that packets 43 and 42 are listed as unreceived. Updated values are:

Highest Received	Previously Received List							
44	43	42	41	40	39	38	37	36
	N	N	Y	Y	Y	Y	Y	Y

4. Packet 39 received. Dropped because it is a duplicate with previously received packet. No update to Highest Received and Previously Received List. Updated values are:

Highest Received	Previously Received List							
44	43	42	41	40	39	38	37	36
	N	N	Y	Y	Y	Y	Y	Y

5. Packet 42 received. Accepted because it is listed as unreceived on the Previously Received List. Packet 42 listed as received. No update to Highest Received because 42 is not higher than 44. Updated values are:

Highest Received	Previously Received List							
44	43	42	41	40	39	38	37	36
	N	Y	Y	Y	Y	Y	Y	Y

Appendix B - Command Assignments

The following is to be added to Table G-1 to cover the new command assignments for IPMI v2.0. (See Appendix G in IPMI v1.5 specification for key.)

Table B-1, Command Number Assignments and Privilege Levels

	section	NetFn	CMD	C	U	O	A
BMC Device and Messaging Commands							
...							
Activate Payload		App	48h	X ⁵			
Deactivate Payload		App	49h	X ⁵			
Get Payload Activation Status		App	4Ah		X		
Get Payload Instance Info		App	4Bh		X		
Set User Payload Access		App	4Ch				
Get User Payload Access		App	4Dh		X		
Get Channel Payload Support		App	4Eh		X		
Get Channel Payload Version		App	4Fh		X		
Get Channel OEM Payload Info		App	50h		X		
unassigned		App	51h	-	-	-	-
Master Write/Read		App	52h			X	
Get Channel Authentication Capabilities		App	53h	p	p	p	p
Get Channel Cipher Suites		App	54h	p	p	p	p
Suspend/Resume Payload Encryption		App	55h		X ⁹		
Set Channel Security Keys		App	56h				X
SOL Commands							
SOL Activating		Transport	20h				
Set SOL Configuration Parameters		Transport	21h				X
Get SOL Configuration Parameters		Transport	22h		X		

5. A session operating at Callback, User, or Operator can only use this command to terminate their own session. An Administrator or system software can use the command to terminate any session.
9. The Suspend/Resume command may be blocked by a configuration option for the particular payload type that forces encryption to be used. In this case, an Admin level command would typically be required to change the configuration.