(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
22 March 2007 (22.03.2007)

PCT

(10) International Publication Number
WO 2007/033207 A2

(51) International Patent Classification:
A63F 9/24 (2006.01)

(21) International Application Number: PCT/US2006/035556

(22) International Filing Date: 11 September 2006 (11.09.2006)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:

60/716,713	12 September 2005 (12.09.2005)	US
11/530,450	8 September 2006 (08.09.2006)	US
11/530,452	8 September 2006 (08.09.2006)	US

(71) Applicant *(for all designated States except US)*: **BALLY GAMING, INC.** [US/US]; 6601 South Bermuda Road, Las Vegas, Nevada 89119 (US).

(72) Inventors; and
(75) Inventors/Applicants *(for US only)*: **ARBOGAST, Chris** [US/US]; Las Vegas, Nevada (US). **GREEN, Travis** [US/US]; Las Vegas, Nevada (US). **JONES, Wil** [US/US]; Las Vegas, Nevada (US). **SHEPHERD, Dale** [US/US]; Las Vegas, Nevada (US). **CADIMA, Ron** [US/US]; Las Vegas, Nevada (US). **BUCKEYNE, Thomas** [US/US]; Las Vegas, Nevada (US). **GREEN, Anthony** [US/US]; 1760 Crystal Stream Avenue, Henderson, Nevada 89012 (US). **PRAVINKUMAR, Patel** [US/US]; 8445 Las Vegas Blvd., South 2062, Las Vegas, Nevada 89123 (US). **CROWDER, Robert, W.** [US/US]; 15 Chateau Whistler Court, Las Vegas, Nevada 89148 (US). **LARSEN, Josh** [US/US]; Las Vegas, Nevada (US).

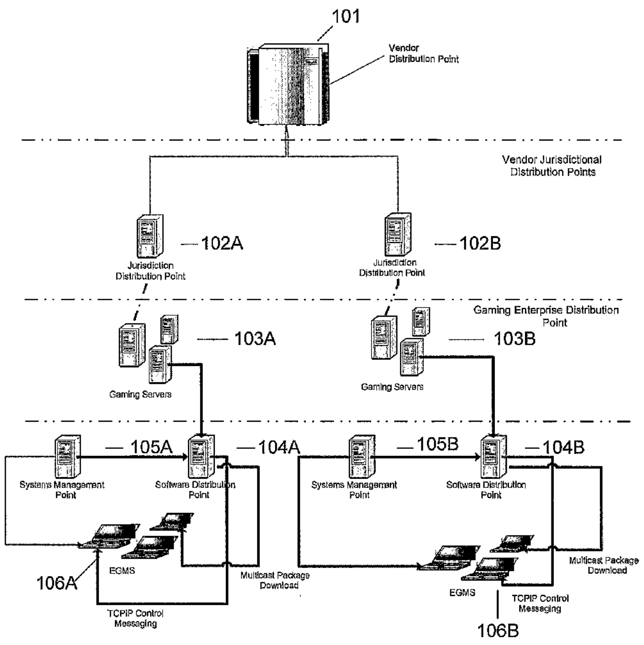
(74) Agents: KOVELMAN, Robert, L. et al.; STEPTOE & JOHNSON, LLP, 1330 Connecticut Avenue, NW, Washington, DC 20036 (US).

(81) Designated States *(unless otherwise indicated, for every kind of national protection available)*: AE, AG, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BW, BY, BZ, CA, CH, CN, CO, CR, CU, CZ, DE, DK, DM, DZ, EC, EE, EG, ES, FI, GB, GD, GE, GH, GM, HN, HR, HU, ID, IL, IN, IS, JP, KE, KG, KM, KN, KP, KR, KZ, LA, LC, LK, LR, LS, LT, LU, LV, LY, MA, MD, MG, MK, MN, MW, MX, MY, MZ, NA, NG, NI, NO, NZ, OM, PG, PH, PL, PT, RO, RS, RU, SC, SD, SE, SG, SK, SL, SM, SV, SY, TJ, TM, TN, TR, TT, TZ, UA, UG, US, UZ, VC, VN, ZA, ZM, ZW.

(84) Designated States *(unless otherwise indicated, for every kind of regional protection available)*: ARIPO (BW, GH, GM, KE, LS, MW, MZ, NA, SD, SL, SZ, TZ, UG, ZM, ZW), Eurasian (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European (AT, BE, BG, CH, CY, CZ, DE, DK, EE, ES, FI,

[Continued on next page]

(54) Title: DOWNLOAD AND CONFIGURATION SYSTEM AND METHOD FOR GAMING MACHINES


101
Vendor Distribution Point
Vendor Jurisdictional Distribution Points
102A
Jurisdiction Distribution Point
102B
Jurisdiction Distribution Point
Gaming Enterprise Distribution Point
103A
Gaming Servers
103B
Gaming Servers
105A
104A
Systems Management Point
Software Distribution Point
105B
104B
Systems Management Point
Software Distribution Point
EGMS
106A
Multicast Package Download
TCPIP Control Messaging
Multicast Package Download
EGMS
TCPIP Control Messaging
106B

(57) Abstract: The system and method allows a casino operator to manage groups of electronic gaming machines (EGMs). Managing new or existing collections (i.e., groups) of gaming devices reduces the effort required to download or configure large numbers of EGMs. For example, new software may be downloaded to groups of EGMs from a central location, and the EGMs may be configured from the central location. Accordingly, this operational efficiency reduces maintenance costs and minimizes EGM downtime due to maintenance or EGM set-up.

FR, GB, GR, HU, IE, IS, IT, LT, LU, LV, MC, NL, PL, PT, RO, SE, SI, SK, TR), OAPI (BF, BJ, CF, CG, CI, CM, GA, GN, GQ, GW, ML, MR, NE, SN, TD, TG).

Published:

— *without international search report and to be republished upon receipt of that report*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

DOWNLOAD AND CONFIGURATION SYSTEM AND METHOD FOR GAMING MACHINES

COPYRIGHT NOTICE

BACKGROUND

[0002] Over the years, casinos have grown in size, grandeur, and amenities in order to attract gambling patrons. Additionally, casinos have attempted to provide gambling patrons with a wide variety of the new and exciting games. Given this demand, gaming machines have grown in sophistication and features in order to captivate and maintain player interest. As a result, casinos are able to provide a wide range and large number of games of chance.

[0003] For example, a casino floor may include thousands of electronic gaming machines (EGMs) that are in communication with and monitored by the casino's gaming network. EGMs provide an enhanced gaming experience with computer graphics, stereo sound, animation, and other features that have been developed to maintain player interest in the game. Furthermore, EGMs may include secondary networked devices such as player tracking devices or enhanced player interfaces (e.g., Bally Gaming's iView^{TM} touch-screen display). Accordingly, there are a large number of EGMs and related components that need to be monitored, maintained, and serviced.

[0004] In early gaming environments, gaming machines were stand-alone devices. Security of the gaming machines was accomplished via physical locks, security protocols, security personnel, physical and video monitoring, and the need to be physically present at a machine to attempt to breach the security of the gaming machine. By the same token, management of the gaming machines required a great deal of personal physical interaction with each gaming machine. The ability to change parameters of the gaming machine also required physical interaction.

[0005] In view of the increased processing power and availability of computing devices, gaming machines have become customizable via electronic communications and remotely controllable. Manufacturers of gaming equipment have taken advantage of the increased functionality of gaming machines by adding additional features to gaming machines, thereby maintaining a player's attention to the gaming machines for longer periods of time increasing minimum bet and bet frequency and speed of play. This, in turn, leads to the player wagering at the gaming machine for longer periods of time, with more money at a faster pace, thereby increasing owner profits.

[0006] The amount of interactivity and data presentation/collection possible with current processor based gaming machines has led to a desire to connect gaming machines in a gaming network. In addition to the gaming machines themselves, a number of devices associated with a gaming machine or with a group of gaming machines may be part of the network. It has become important for the devices within a gaming machine or cabinet to be aware of each other and to be able to communicate to a control server. Not only is the presence or absence of a network device important, but also the physical location of the device and the ability to associate devices within a particular gaming machine has become a necessary component of a gaming network.

[0007] Currently, casino operators use manual methods to alter content or to reconfigure EGMs and/or other secondary networked devices. For example, a casino employee would need to physically swap out an EPROM to change game content or the employee would need to access an attendant menu on the EGM to alter game configurations. Given the large number of machines and networked devices, this process is a time-consuming and costly process not only in terms of operating and/or maintenance costs, but also in terms of lost profits due to extended downtime for the EGMs. Similarly, existing approaches for software updates or downloads for EGMs are labor-intensive and costly as the EGMs. For example, a technician typically needs to travel to the gaming machine in order to replace existing software package media (e.g., EPROMs, CD-ROM's, Compact Flash, etc.) with new software package media. Furthermore, the software package update process may require that the EGM be disabled hours in advance to prevent any players from using the EGM when the technician is ready to perform software package changes. Alternatively, EGMs may be disabled prior to software package updates, but the technician must periodically check to ensure that the EGM(s) are not being used by a player. Additionally, technicians may need to be supervised during the process of software package installation as the technician has access to critical areas of the EGM required for configuration or of those areas of containing cash.

[0008] The process of transferring packages to an EGM over a network may require a significant amount of network bandwidth during the transfer period. Typical transfer mechanisms provide point-to-point transfer where a Software Distribution Point (SDP) will transfer to a single EGM until the transfer is complete, and then the SDP may transfer to another EGM. When hundreds or thousands of EGM's require packages to be transferred there may be an unacceptable extended period of high bandwidth usage, since the transfers must occur sequentially.

[0009] Additionally, installing packages on an EGM can require verification that dependent software packages and hardware components are available within the EGM. This is typically a manual process that prone to human interpretation and human error.

[0010] Accordingly, there remains a need to provide a system for updating and configuring EGMS and other networked components.

SUMMARY

[0011] Briefly, and in general terms, various embodiments are directed to management of gaming devices over a network. The system allows a casino operator to manage groups of electronic gaming machines (EGMs). Managing new or existing collections (i.e., groups) of gaming devices reduces the effort required to download or configure large numbers of EGMs. For example, new software may be downloaded to groups of EGMs from a central location, and the EGMs may be configured from the central location. Accordingly, this operational efficiency reduces maintenance costs and minimizes EGM downtime due to maintenance or EGM set-up.

[0012] In one embodiment, EGMs can be updated over the network via a multicast over TCP/IP or some other suitable mechanism. A package containing game information and/or configuration information is created at a server. The package is delivered to an EGM over the network. The packages can be downloaded via a background thread and may take place while the EGM is enabled or disabled or even when a player is actively playing the EGM.

[0013] The system includes a scheme for EGM's to report corrupted or missing packets from the transfer, which the SDP will use to resend the packets. The SDP will use the multicast protocol to resend the packets, potentially reducing the network bandwidth used during error recovery.

[0014] The system includes a package validation method that uses digital signatures. A digital signature is provided with each package, which an EGM can use to validate the package and authenticate the origin of the package. The EGM will not install or use a package that does not pass validation.

[0015] The system includes technology necessary for the System Management Point (SMP) to manage package dependencies (dependencies such as a new package-A that requires package-B and package-C to operate). This technology can be used to confirm that new package-A dependencies are addressed before requesting the EGM to transfer a new package. Additionally, the SMP can determine the dependencies and request that those packages also be transferred to the EGM.

[0016] The system includes technology necessary for the SMP to determine if a package's dependent hardware components are available on the EGM. The concept is similar to the package dependencies; however the SMP can not automatically transfer hardware components to an EGM to satisfy package dependencies. If a package's required

hardware dependencies are available on the EGM, then the package transfer to the EGM will be permitted; otherwise, the package transfer will not be permitted.

[0017] According to one method, the system may designate that a collection of gaming machines should have a particular status at a given time. Changes or status may be applied to different collections of gaming machines. The network system allows a casino operator to define changes to software inventory (i.e., via download) and to schedule configuration changes. For example, a casino operator can define default payouts and denominations, schedule recurring overrides for weekends, and schedule a one-time override for a holiday or casino promotion. Accordingly, these changes can occur without further operator interaction. As a result, casino management does not need to be in attendance every time the slot floor assumes a new configuration.

[0018] Moreover, the configuration changes are communicated during the background operation of the EGM without affecting gameplay, and the changes are applied at a designated convenient time (e.g., a predetermined period of time after the credit meter reaches zero credits). Accordingly, the EGM does not need to be placed in an inactive state prior to downloading configuration changes.

[0019] Furthermore, various methods of assignment conflict analysis and resolution are disclosed herein. Typically, conflicts will only occur for assignments of the same type (e.g., download and configuration). Generally, download/configuration conflicts are avoided by running the download process before the configuration process with the exception that data related to host and owner information will supersede other assignments. Alternatively, conflicts may arise as a result of an EGM being a member of different collections where membership may vary depending on the moment in time. For example, switching an EGM from a 5 cent denomination to a 25 cent denomination may switch the membership of the EGM to another collection (e.g., all 25 cent EGMs). Accordingly, the network system includes various methods to resolve situations where EGMs are given improper settings (e.g., the system will filter out settings that the EGM does not support). As a result, these methods provide consistent procedures as to how the network host configures EGMs when more than one assignment applies.

[0020] In another method, methods of pre-configuring EGMs are disclosed herein. In one method, the network system uses option templates to support pre-configuration because a particular EGM or game theme within an EGM may support a large number and wide variety of options. For example, option definition templates such as Combo Option templates or

Option Group templates may be used to define the configuration of new content before it is downloaded to an EGM. Additionally, a casino operator may schedule the download of a new game theme during off-hours and have the network host configure the new game theme as soon as installation is completes without requiring operator intervention.

[0021] Methods of automatic downloading and configuration of EGMs are also disclosed. In one method, the network system provides a method of recognizing when an EGM needs data downloads or configuration, and the network coordinates these activities to avoid conflicts. For example, in one method, attempts to configure an EGM will be prevented until downloads for the EGM are completed. In another method, the network host automatically restores data modules and configures an EGM if it has been RAM-cleared or has been offline. Accordingly, an operator can monitor and manage a group of EGMs from a single terminal, thereby eliminating the need for slot technicians to collect configuration data and to manually reconfigure each EGM.

[0022] In yet another method, methods of simultaneous download to multiple EGMs are disclosed herein. The network system distributes (i.e., downloads) data packages through the use multicast sessions, where the EGMs request all or part of a data package (as directed by the download server). This method may also include a lossless transport mechanism and an IP multicast as the distribution mechanism. The multicast distribution mechanism conserves network bandwidth, can be throttled at the host, and prevents low priority activities (e.g., download) from interfering with critical communications (e.g., vouchers or events).

[0023] Another method is directed to minimizing memory storage consumption associated with package downloads to EGMs. In this method, the host downloads a package (e.g., BLOB (Binary List of Bytes)) to the EGM. The package is then authenticated and may then be installed immediately or at a later scheduled time. Once the contents of the package are installed on the EGM, the package may optionally be removed from the EGM. For example, portions of package may be removed by overwriting the package with new packages. The package's contents are tracked by the resulting versioned modules that have been installed on the EGM.

[0024] Other features and advantages will become apparent from the following detailed description, taken in conjunction with the accompanying drawings, which illustrate by way of example, the features of the various embodiments.

BRIEF DESCRIPTION OF THE DRAWINGS

[0025] Figure 1 illustrates an embodiment of a gaming network that may be used with the system.

[0026] Figure 2 illustrates an overview of download interaction

[0027] Figure 3 illustrates an overview of EGM Startup message flow.

[0028] Figure 4 depicts the logic flow of /dev/download driver initialization.

[0029] Figure 5 is a flow diagram illustrating a routine to validate download requests and pass control to the proper handler function.

[0030] Figure 6 is a flow diagram illustrating the add package logic of the download driver.

[0031] Figure 7 is a flow diagram illustrating the download control process addPackage logic.

[0032] Figure 8 illustrates the message flow for the download receiver addPackage process.

[0033] Figure 9 illustrates the logic flow for the DR.

[0034] Figure 10 is a flow diagram illustrating the pre-requisite process.

[0035] Figure 11 illustrates the logic flow for this process.

[0036] Figure 12 is an overview of how the package is installed on the EGM.

DESCRIPTION OF EMBODIMENTS

[0037] Various network systems and methods for managing networked gaming machines are disclosed herein. The system supports data downloads and configuration of gaming machines such as, but not limited to, electronic gaming machines (EGMs). With the network system, a casino operator may define a collection of EGMs, assign modules to one or more collections of EGMs, assign configuration changes to one or more collections of EGMs, and schedule all assignments.

[0038] The system permits some or all of the software that is to be run on an EGM to be centrally stored in a software distribution library located on one or more servers. The software can be sent over a network (such as an Ethernet network) as desired for initial set up of an EGM, as updating of an EGM, game replacement, configuration, or any other desired purpose.

[0039] The system utilizes multicasting to download software packages to the EGMs in the background, even while full game play is ongoing. The server does a single multicast transfer of one or more packages to all target machines. Each EGM tracks the packages and notes missing, corrupted, or dropped packets. In one embodiment, the server receives requests from all EGMs for packets to resend and again multicasts them to all EGMs, regardless of which EGM requested which packet. The individual EGMs accept needed missing packets and ignore other re-broadcast packets. It should be noted that the system is not limited to multicast transmissions but may use HTTP, HTTPS, FTP, SFTP, and other transmission protocols.

[0040] It should be noted that the term EGM is intended to encompass any type of gaming machine, including hand-held devices used as gaming machines such as cellular based devices (e.g. phones), PDAs, or the like. The EGM can be represented by any network node that can implement a game and is not limited to cabinet based machines. The system has equal applicability to gaming machines implemented as part of video gaming consoles or handheld or other portable devices. In one embodiment, a geo-location device in the handheld or portable gaming device may be used to locate a specific player for regulatory and other purposes. Geo-location techniques that can be used include by way of example, and not by way of limitation, IP address lookup, GPS, cell phone tower location, cell ID, known Wireless Access Point location, Wi-Fi connection used, phone number, physical wire or port on client device, or by middle tier or backend server accessed. In one embodiment, GPS and biometric devices are built within a player's client device, which in one embodiment, comprises a player's own personal computing device, or provided by the casino as an add-on device using USB, Bluetooth, IRDA, serial or other interface to the hardware to enable jurisdictionally compliant gaming, ensuring the location of play and the identity of the player. In another embodiment, the casino provides an entire personal computing device with these devices built in, such as a tablet type computing device, PDA, cell phone or other type of computing device capable of playing system games.

[0041] An embodiment of a network that may be used with the system is illustrated in Figure 1. The software distribution network consists of a top level vender distribution point 101 that contains all packages for all jurisdictions, one or more Jurisdiction distribution points 102A and 102B that contain regulator approved production signed packages used within that jurisdiction or sub-jurisdiction, one or more Software Management Points 103A and 103B to schedule and control the downloading of packages to the EGM and a one or more Software Distribution Points 104A and 104B that contain regulator approved production signed packages only used in the gaming establishment that it supports. The Software Distribution Points (SDPs) 104A and 104B can communicate with Systems Management Points (SMPs) 105A and 105B, respectively as well as directly to one or more EGMs 106A and 106B. (The communication between SDP and SMP is optional. A benefit is to permit the SMP and the user access to the catalogue of software packages available on the SDP). The system allows for rapid and secure distribution of new games and OS's from a centralized point. It makes it possible to update and modify existing gaming machines with fixes and updates to programs as well as providing modifications to such files as screen images, video, sound, pay tables and other EGM control and support files. It provides complete control of gaming machines from a centralized control and distribution point and can minimize the need and delay of human intervention at the EGM.

[0042] In one embodiment, a 1 GB compact flash card will initially be used as the storage media in the EGMs for OS and download package storage. A second compact flash will be used for the storage of the production game as it exists today. The OS compact flash will be partitioned into production, backup, and download areas. The production area contains the active OS and game management software used to allow the playing of games on the EGM. This part of storage will have all the file integrity checking and validation performed on it. The other areas will be used to store backup software and data, new download packages, installation of the new packages and saving of NVRAM, counters, random number data and other secure information in an encrypted data format. Software downloads will be scheduled by gaming personnel via the SMPs. The SMP notifies the individual gaming machines they have been scheduled to receive a download package from a specific vendor Software Distribution Point (SDP). The EGM shall then request the SDP to start sending the download package to it.

[0043] The system solves the network bandwidth usage problem by using a multicast network protocol, which the SDP uses to broadcast the transfer to multiple EGM's simultaneously. The SDP will send a single transfer to a defined set of EGM's, thereby using a fraction of the network bandwidth when compared to the point-to-point strategy. If there are X groups of EGM's, with each group consisting of Y EGM's, then the following calculations provide a general sense of reduced network bandwidth usage:

[0044]	Multicast:	X * transferSize
[0045]	Point-to-point:	(X * Y) * transferSize
[0046]	--------------	---------------------------
[0047]	Gross Improvement:	1/Y network usage with multicast.

[0048] The system includes a recovery scheme to accommodate transmission error or reception error of the package. This recovery scheme allows each EGM to report missing pieces (packets) where either not received or received with errors (data corruption). The SDP logic can accumulate a list of missing packets and the corresponding EGM's, and use the multicast protocol to resend the missing packets.

[0049] The specific packages that are transferred are specified by the EGM. The EGM makes requests to the SDP using a point-to-point protocol. The SDP may not service these requests immediately, so the EGM is permitted to operate while waiting for the transfer and during the transfer. The EGM is responsible for logging transfers and checking the validity of the completed transfer. This log can be used to reconcile the package content of the EGM at any given time. Additionally, the SDP will log transfer activity and provide a means to cross check the package content of an EGM.

[0050] An EGM validates transferred packages using a digital signature. This digital signature provides authentication of the package itself regardless of the specific SDP server that provided it, and is used to determine if the package incurred any transmissions errors. If an EGM is unable to validate a package, then the package will not be installed or otherwise used.

[0051] The system includes technology necessary to determine if an EGM contains packages necessary to use a new package (package dependencies). If a given package-A requires additional packages, package-B and package-C, to operate on the EGM, the dependencies can be determined from a package header contained within package-A. The SMP can then determine if the EGM already contains the dependent packages, package-B and package-C, before requesting that package-A be transferred to the EGM. Similarly, the SMP can automatically select the dependent packages to be transferred in addition to the original package-A, thus fulfilling the dependencies of package-A. This aspect of the system provides a level of assurance that the packages transferred can actually be used by the EGM.

[0052] The system includes technology necessary to determine if a package's dependent hardware components (components) are available on the EGM. If package-X requires component-Y and component-Z, the SMP can read the components of the EGM and use this information for component dependency checking. If a package's required component dependencies are available on the EGM, then the package transfer to the EGM will be permitted; otherwise, the package transfer will not be permitted. This aspect of the system provides a level of assurance that the packages transferred can actually be used by the EGM.

[0053] The ability to transfer software directly to the EGM provides additional alternatives when installing a new EGM on the casino floor. An EGM that is manufactured and placed directly on the casino floor can be initially loaded with a boot-strap media (EPROM, CD-ROM, Compact Flash, etc.) that can boot the EGM and prepare communications before requesting new packages to install. Two boot-strap methods may be used to transfer software to an EGM. Both methods require minimal configuration of an EGM before the boot-strap process can be completed. Minimal configuration consists of the following:

[0054] 1) A unique network IP Address for the EGM. This can be provided by a DHCP system, or assigned to the EGM via operator configuration.

[0055] 2) A unique EGM identifier that is derived from the EGM hardware (such as network card MAC address) or assigned to the EGM via operator configuration.

[0056] 3) An address of the SDP network server to make transfer requests of. This can be a hard-coded default address that may be overridden via operator configuration.

[0057] The simple boot-strap mode requires that EGM is also configured with the necessary options to locate the SMP network server. This can be a hard-coded default address that may be overridden via operator configuration. This address is used by the EGM to locate the SMP server, which can then assign new packages to the EGM for transfer. In one embodiment, the EGM and its associated peripherals can be identified using a scheme such as is described in U. S. Patent Application 11/319,034 entitled "Device Identification" and incorporated in its entirety herein by reference. In this embodiment, during start-up, a device sends its MAC address out on the network. A local switch collects MAC and IP addresses for the devices connected to it. Periodically, the switch transmits raw Ethernet frames, USB packets, or TCP packets containing tables of devices and associated MAC/IP addresses. When a device receives information about another device, the device may attempt communication with that device. First, a verification procedure is used to validate the devices. Subsequently, communication is possible between the devices.

[0058] The advanced boot-strap mode involves the EGM requesting a default boot-strap package from the SDP. The package name for this boot-strap package is predefined and known to both the EGM and the SDP. This boot-strap package contains configuration files that identify the default software packages and configuration data. The boot-strap package can be customized for the specific installation (casino) and stored on the SDP. This way all EGM's will request the predefined boot-strap package, the SDP will transfer the boot-strap package to the EGM, and the EGM will process the boot-strap package. When the EGM processes the boot-strap package, the EGM can read the SMP address, default EGM OS package, and other packages such as default peripheral firmware packages. At this point the EGM can request these packages from the SDP, and the SDP will begin transferring the default packages to the EGM, ultimately ending with an EGM ready for configuration.

[0059] Figure 2 illustrates an overview of download interaction between the System Management Point 105, EGM 106, and Software Distribution Point 104. The design consists of a download device driver which provides the central control logic and logic. It interfaces with the Download Class (which may be G2S or any other protocol) and a download control thread. The download control thread controls requesting and downloading packages from the download distribution point server.

[0060] In operation, the System Management Point 105 sends package commands and requests to the Download Control 206 of EGM 106 to tell it to add and delete packages, install packages and request information about packages and installed modules. The

Download Control 206 will filter these requests and pass then to the Download Driver 207 for processing. Requests to add packages are then given to the Download Receiver Process 208 which then opens communications with the Software Distribution Point 104. The Download Receiver Process 208 receives the package data from the Software Distribution Point 104 via a multicast (or any other defined transmission protocol) link 204 in multiple packets and assembles the package in compact flash. Once all packets of the package have been received, the package data is verified using a SHA-1 verification string passed in the package header. The appropriate status and package state information shall be passed back to the Download Driver 207 which passes it to the Download Control 206 for logging and passing back to the System Management Point.

[0061] Download Driver 207

[0062] The download driver is installed when the EGM system is started. It's primary functions are to:

[0063] 1 - Initialize package and control information as stored on the compact flash.

[0064] 2 - Process commands to add packages and install rules from the Download Control 206.

[0065] 3 - Supply the Download Control 206 with list and status information and download and install events.

[0066] 4 - Queue add package requests for the Download Receiver process 208 and pass them to the Download Receiver 208 via a read 211 to the driver.

[0067] 5 - Update package the status and error conditions of packages and install rules and pass them back to the Download Control Process 206.

[0068] Download Control Thread 206

[0069] The Download Control Process 206 is either a thread running within the game manager context or a standalone process when there is no game manager running on the system. It's primary tasks are:

[0070] 1 - Receives download commands

[0071] 2 - Passes the commands that it will not process on to the download driver 207 for delivery to the Download Receiver process 208 and the Download Installer process 209..

[0072] 3 - Receive download and install status from the driver and log in the appropriate log files on the EGM 106.

[0073] 4 - Supply the log information and package, install rule and module lists to the System Management Point 105 when requested.

[0074] Download Receiver Process 208

[0075] The Download Receiver Process 208 is responsible for communications between the EGM 106 and the Software Distribution Point 104. It's primary functions are:

[0076] 1 - Receive add package requests from the Download Driver 207.

[0077] 2 - Open a TCPIP link to the Software Distribution Point 104 and request the package defined within the add package request be sent to the EGM 106.

[0078] 3 - Open a multicast communications port specified by the Software Distribution Point 104 and receive the requested package in multiple packet format.

[0079] 4 - Assemble the packets of the package into a single file.

[0080] 5 - Request any missed or missing packets be resent.

[0081] 6 - Verify the package data by calculating a SHA-1 value and matching that against the SHA-1 validation string passed in the package header.

[0082] 7 - Send back status and error information of the package receive process to the Download Control Process 206 via the Download Driver 207.

[0083] Download Package Installer Process 209

[0084] The Download Package Installer Process 209 is responsible for installing the packages that are downloaded from the Software Distribution Point 104. It is notified of when to start installing a downloaded package when a set install rule command is sent from the Systems Management Point 105. It receives this command via a read to the Download Driver 207. The basic functions of the Download Package Installer Process 109 are:

[0085] 1 - Parse the set install rules to determine how to process the package.

[0086] 2 - Disable the EGM 106 based on the instructions in the set install rule command.

[0087] 3 - Install the package using the command file sent as part of the download package.

[0088] 4 - Clear NVRAM as instructed by the install rules.

[0089] 5 - Reboot the EGM 106 as instructed by the set install rules.

[0090] 6 - Pass back status and error information via the Download Driver 207.

[0091] EGM Initialization

[0092] Figure 3 illustrates an overview of EGM startup message flow. EGM 106 sends a message 302 to SDP 104 requesting a package. SDP 104 replies with a message 303 indicating package size and the multicast IP/Socket that will be used for transmission. Message 304 from SDP 104 to EGM 106 is the package data itself. If necessary, EGM 106 can request missed packets by sending a message 305.

[0093] As noted above, the system uses multicasting to send packages to multiple EGMs. When packets are missed and requested by an EGM, SDP 104 collects a number of requests and sends out multicasts of all missed packets to all of the EGMs. If a particular EGM does not need one of these re-broadcast packets, it is ignored. Otherwise, the EGM accepts the re-broadcast packet and uses it to complete the package transmission.

[0094] Download Driver

[0095] In the interface between the download driver 207 and the download processes, all set status, set error information and command information is passed to the driver 207 via an IOCTL call. The processes use the read interface to get processing instructions from the driver 207. The Download Receiver 208 uses the read interface to get add package requests, the Download Installer Process 209 uses the read to obtain set install rule requests, and the Download Control Process 206 uses the read interface to receive the status and error information from the Receiver and Install processes.

[0096] /dev/download is a loadable module, "device driver", that is loaded as the system comes up. (It is a Linux loadable module in one embodiment). DI_Init_module() is the first routine to gain control when the module is loaded. It will:

[0097] 1 - Allocate the download buffer used to store Download Class messages. (e.g. the G2S or other protocol)

[0098] 2 - Register the download support as a Linux kernel device.

[0099] 3 - Initialize the read queues fro the processes that communicate with the driver.

[00100] 4 - Initialize the storage area for package and set install rules that may be received.

[00101] Once the driver has been installed and initialized, all further actions will controlled by the ioctl and read function entry points into the driver. The ioctl entry point, DL_ioctl(), will process all the System Management Point 105 requests passed through the Download Control Process received from the download class. The DL_read() entry point services all the read requests from:

[00102] · The Download Control Process 206 for packages status and error,

[00103] · The Download Receiver process 208 to pass add package requests.

[00104] · The Download Installer 209 to receive set install rule request.

[00105] Figure 4 depicts the logic flow of /dev/download driver initialization. At step 401 the Read Queue areas and semaphores are initialized. At step 402 the package is initialized and the Install Rule Data Structures are set. At step 403 the driver is registered with the system and at step 404 the system returns.

[00106] Once the /dev/download driver is installed and initialized, the dl_ioctl() driver 207 serves as the interface to the Download Control Process 206, and the Receiver 208 and Install 209 processes. All requests from System Management Point 105 are passed through Download Control Process 206 and either passed onto the driver 207 to be processed, or in the case of module and log requests, processed within the Download Control Process 206 itself. The dl_ioctl() is a command ID to determine what to do with the specific request. Figure 5 is a flow diagram illustrating a routine to validate download requests and pass control to the proper handler function.

[00107] At step 501 the particular switch that is set on the Command ID is examined so that the request from the SMP 105 can be routed to the correct handler. Add and delete Package commands are provide to DL_Packages 502. Set and Delete Install rules are passed to DL_Install 503. Set Status, Retrieve Status, Retrieve List commands are sent to DL_Status 504. Register processes are sent to block 505 where the Pid of the Process being registered is saved. Unknown commands return an error event at 506.

[00108] <u>addPackage Processing Logic</u>

[00109] The addPackage command from the Download class is processed by the Download Control Process 206, the download driver 207 and the download receiver process

208. Breaking up the processing into these three areas helps to maintain isolation between specific function for easily adapting the code to support operation both with any protocols as well as providing the capability to integrate different package download protocols and requirements that may arise in the future. Figure 6 illustrates the download driver 207 addPackage logic. The addPackage logic checks to see if the package already exists on the EGM and if there is enough room to store the package information. If so, it will send the package information to the Download receiver's driver read queue and the Download control driver's read queue and post the read semaphores if a read is outstanding.

[00110] At step 601 there is an addPackage request. At step 602 it is determined if the requested package exists. If no, the system returns a package exists error at step 603. Otherwise the system proceeds to step 604 to determine if the maximum number of packages is already queued. If yes, a return queue full error is returned at step 605. Otherwise the system proceeds to step 606 and adds addpackage information into the package table.

[00111] At step 607 addPackage is placed in the receiver read queue. At step 608 it is determined if there is a receiver read outstanding. If not, the system returns normal status at step 609. Otherwise, the system posts a receiver read sleep semaphore at step 610. At step 611 the addPackage status is placed in the control read queue. At step 612 it is determined if there is a control read outstanding. If no, the system returns normal status at steps 609. If yes, the system posts a control read sleep semaphore at step 613 and returns normal status at step 609.

[00112] Download Control Process addPackage Logic

[00113] The download control process 206 has a thread that performs reads from the download driver 207. When the read is satisfied for an addPackage or package status, the operation of Figure 7 is followed. The package information is written to the /Packages/Packages directory using the package ID as the file name. A log entry is also created and written to the package log in the /Packages/Logs directory. Finally a package status message is created and sent back for delivery to the System Management Point.

[00114] At step 701 the addPackage is read from the driver. At step 702 the addPackage information is written to disk. At step 703 it is determined if the write operation was successful. If not, an error event is created and sent at step 704. Otherwise, a package log entry is created and written at step 705. At step 706 it is determined if the log write was successful. If not, an error event is sent at step 707. If the log write is successful, or after

step 707, a package status message is created and sent at step 708. At step 709 a read is issued to the download driver.

[00115] Download Receiver addPackage

[00116] The Download receiver is responsible for downloading the package identified in the addPackage message from the Software Distribution Point 104. Figure 8 illustrates the message flow for accomplishing this task. The message flow is between the SMP 105 and EGM 106, and between EGM 106 and SDP 104.

[00117] Within the EGM 106, the Download Receiver (DR) process 208 is the one that communicates with the Software Distribution Server. The addPackage command 801 contains the IP and socket address of the Distribution Server to be contacted. The DR 208 opens the link to the SDP 104 and requests 806 the package identified within the addPackage command be sent to it. The SDP 104 responds 807 with a message containing packet size, timeout values, the size of the package, and the multicast port to receive the data on. The DR 208 sends download initiate status 802 and download progress messages 803 to SMP 105.

[00118] As data packets are downloaded 808, the DR 208 keeps track of the packets sent and will request a resend 809 for any packets not received. Those packets are resent 810 as part of a multicast. After all packets have been received 804, a SHA-1 value will be calculated for the data portion of the package and compared 805 against the validation string sent as part of the package.

[00119] Figure 9 illustrates the logic flow for the DR 208. At step 901 the DR read is initiated. At step 902 a link is opened to the SDP 104. At step 903 it is determined if a link is established. If not, a package error is sent to the driver at step 904 and a read is issued to the driver at step 905. Otherwise a request package is sent to the SDP at step 906.

[00120] The response from the SDP is read at step 907. At step 908 a multicast socket is opened. Ata step 909 the status is sent to the DR via ioctl. At step 910 the DR reads from the multicast socket. At step 911 it is determined if there is a read timeout condition. If not, at step 912 the packet data is saved at a specified offset and save sequence. Otherwise it is determined at step 913 if there are any missed packets. If yes, then at step 914 a resend packet request is sent to the SDP and the system returns to step 909.

[00121] Otherwise at step 915 the downloaded package is validated (e.g. using SHA-1). At step 916 it is determined if the data is validated. If not, a package error status message is

sent to the download driver at step 917. Otherwise a package validated status is sent to the download driver at step 918 and the system returns to step 905.

[00122] deletePackage

[00123] The deletePackage command is processed by the Download Control 206 and Receiver 208 processes and by the Download Driver 207. When the download driver 207 receives a deletePackage request, if the status of the package is not validated or there is an error, the status is reset to delete pending and the delete package command is sent to the download receiver 208. The download receiver process 208 should then delete any files it has created and terminate any download activity for the file. When the cleanup is complete, it will send a deleted status back to the download driver 207 and resume its read from the download driver 207.

[00124] When the download driver 207 receives a package deleted status, or if the original status of the package was validated or error, the download driver 207 frees the package data save area and sends a deletePackage complete status to the Download controller process 206. The download controller process 206 logs the new status in the package log and deletes the package status file from the /Packages/Packages directory. It then sends the delete complete status back to the SMP 105.

[00125] setInstallRule

[00126] The setInstallRule Download Class command is processed by the Download Control process 206, the Download Installer Process 209 and the Download Driver 207. The Download Control process 206 maintains the updated status of the setInstallRule on disk and in the install rules log file, as well as sending the status back to the SMP 105 via messages. The download driver 207 acts as the interface between the Download Control Process 206 and the Download install process 209. It will also validate the package does not already exist and that there is enough room to process the install rules.

[00127] Download Installer Process – setInstallRule

[00128] Parsing of the setInstallRule command and the pre-requisite conditions contained in the package header is the first step in installing a download package. The package header is examined to extract the information as to what hardware requirements the package to be installed needs as well as any software that needs to be on the EGM in order to support the package. With respect to software prerequisites, these can refer to already installed software,

or software contained in other packages that have been downloaded and are ready to install. Figure 10 is a flow diagram illustrating the pre-requisite process.

[00129] At step 1001 the validation process begins. At step 1002 it is determined if the requisite hardware is present for the download package. If so, a table of installed hardware data is build at step 1003. At step 1004 the loop of hardware prerequisites is read from the package header. At step 1005 it is determined if the prerequisites are in fact in installed in the hardware table. If so it is determined at step 1006 if all the prerequisites are met. If so, return success at step 1008. If not, return to step 1004.

[00130] If the prerequisites are not installed in the table at step 1005, an install error is logged at step 1009. At step 1010 the install status is sent to the SMP 105 and an error is returned at step 1011.

[00131] If the hardware prerequisites are not present at step 1002, it is determined at step 1007 if the software prerequisites are present. If no prerequisites are specified, return success at step 1008 If yes at step 1007, build a table of required software at step 1012. At step 1013 build a table of available modules from the installed module inventory. At step 1014 build a table of modules from available packages. At step 1015 loop through the prerequisite table.

[00132] At step 1016 determine if the prerequisites are found in the installed modules. If the prerequisite isn't satisfied by the installed modules, step 1019 determines if the prerequisite may be satisfied by packages that have not yet been installed If the prerequisite can't be satisfied, log install error at step 1009. Otherwise remove the prerequisite from the table at step 1017. At step 1018 determine if all prerequisites are satisfied. If yes return success at step 1008. If not, return to step 1014.

[00133] If 1016 is no, determine if the prerequisite is in the package table at step 1019. If not, log install error at step 1009. If so, determine at step 1020 if the package prerequisites are in the prerequisite table. If yes, return to step 1015. If not, add package modules to the prerequisite table at step 1021 and return to step 1015.

[00134] Once all the prerequisites have been met for the all the required packages, the package install rules will be processed. The install rules contain information on when to disable the machine to prevent game playing and if a time delay is required after disabling the game, what trigger to use to start the installation process, and whether the host needs to authorize the install once all other conditions have been met. Figure 11 illustrates the logic flow for this process.

[00135] At step 1101 the DL_Installer() routine is initiated. At step 1102 it is determined if the time window for the install is met. If not, the system waits for the install period at step 1103. Otherwise at step 1104 it is determined if the disable conditions are met. If so, disable the coin and bill collectors at step 1105. Oherwise determine if disable should be immediate at step 1106. if not determine if there should be disable none at step 1107. If no, determine if disable zero credit is true at step 1108. If note return error and exit at step 1109.

[00136] If true at step 1108 determine if the disable wait time has expired at step 1110. If not wait for the disable time at step 1111. For true at 1106,, 1108, 1110, or after step 1111, disable the coin and bill acceptors at step 1105. After step 1105 or if true at step 1107, determine if automatic initiation is true at step 1112. If not, determine if initiate none is true at step 1113. If not determine if initiate operator is strue at step 1114. If not, error exit at step 1116. Otherwise display waiting for initiate install at 1115.

[00137] For true at 1112 and 1113, or after 1115, determine if host authorization is required at step 1117. If no, start installing package at step 1118. Otherwise determine if host authorization is received at step 1119. If not, wait for host authorization at 1120. Otherwise start installing the package at step 1118.

[00138] Package Installation – Installing the package contents

[00139] Once all of the install requirements and prerequisites have been satisfied, the package is ready to be installed. Once the installation process starts, it should not be cancelled or interrupted.

[00140] The package downloaded is made up of at least three distinct parts.

[00141] 1 - A header which contains the following information:

[00142] i. The module names, descriptions and release version number.

[00143] ii. The files and their offsets into the data package.

[00144] iii. The name of the command procedure used to install the package and its offset into the data package.

[00145] 2 - A SHA-1 validation string used to validate the data portion of the package.

[00146] 3 - A data portion of the package containing the installation procedure file and all the files to be installed for this package. This data partition may be in a compressed or uncompressed format. Currently compressed formats that are supported are gzip and bzip2.

Other compression techniques can be accommodated with the actual package data itself and may be processed by the installation command file sent in the package data.

[00147] Figure 12 is an overview of how the package is installed on the EGM. At step 1201 the package install process is initiated. At step 1202 data is copied from the package to a file. At step 1203 it is determined if the package data is compressed. If yes, the system uncompresses the file at step 1204. Otherwise at step 1205 loop through the file definition in the package header. At step 1206 create an individual file from the file portion of the package file. Determine if all files have been created at step 1207. If not, return to 1205.

[00148] Otherwise execute the install file command at step 1208. At step 1209 determine if a reboot is specified in InstallRule. If not, then remove setInstallRule and log and send status at step 1210 and return success at step 1211. Otherwise determine if the prerequisite packages are installed at step 1212. If not, log error and send status to SMP at step 1213 and return error at step 1214. Othewise set NVRAM clear conditions and delete install rules at step 1215, and reboot the EGM at step 1216.

[00149] Alternate Package Install Handling

[00150] With the introduction of new file validation methodologies and the transition to G2S, the system contemplates an alternate embodiment for download package install handling. The current Download interface can be modified to present the Download Installer code with G2S like commands. That is, the SetInstallRule commands may be changed changed into setScript commands for processing by the Download Installed. Also, the getScriptList and GetScriptStatus commands will map the getInstallRuleStatus and getInstallRuleList commands.

[00151] It is assumed that all the commands dealing with download logs will be handled in the G2S support code and will not be a part of the Download support.

[00152] · CURL will be used to provide the support for downloading packages via HTTPS, SFTP, FTP, HTTP, etc. For any multicast protocol, a locally developed protocol may be used.

[00153] This alternate embodiment is based on the following commands:

[00154] 1. A separate thread is used to issue reads to the download driver to receive setScript, deleteScript, authorizeScript commands.

[00155] 2. A table of scripts is maintained. Each entry in the script table will point to the next entry in the script table. A global pointer will be used to point to the first script in the table. The table will be arranged in a FIFO queue and the scripts are processed in the order in which the setScript commands install time frames are specified. If an authorizeScript command is received before it's setScript command, it is rejected and an error event sent back to the server sending the authorization command.

[00156] The script table may be maintained in both memory and on disk. The status of the script entry will be update on disk before the in memory copy.

[00157] 3. When any of the script commands are received the following will happen:

[00158] · setScript

[00159] . If no setScript record exists for this script, create and initialize script record with a state of waiting to process.

[00160] . If other script records exist, place this into the process queue according to its installation start time frame value.

[00161] . If no other scripts in the process queue, place it at the beginning of the process queue.

[00162] . If script waiting for start install time frame and has a start install time frame that is after the script just received, place the already active script back into the process queue and set the new script to waiting for start time frame

[00163] . If machine is in disable state and currently processing another script, just place the script into the script queue on disk.

[00164] . deleteScript

[00165] . If no script record for the specified script, return error, no script present

[00166] . If script record in process queue, remove from process queue and send script deleted event.

[00167] . If script is processing, and process state is installing, send event script installing, not deleted.

[00168] • If script is processing and not in installing state, send event script canceled, delete script record and reset states. If script waiting in script queue, start processing next script.

[00169] • authorizeScript

[00170] Multiple hosts may be required to authorize a script to proceed with installation. Must maintain a list of authorizing hosts and set their authorization state when received. Installation can not proceed until all hosts authorize it.

[00171] • If no script record exists for specified script, reject authorize command and send back an error event.

[00172] • If processing script, set script state to what is specified in the command for the particular host specified in the authorize command.

[00173] • If not processing script, set authorization state to what is specified in command for the specified host.

[00174] <u>Processing setScript Command</u>

[00175] When a setScript command enters the processing state, the order in which things occur are:

[00176] 1) Check dependencies: hardware and modules. Module dependencies can be satisfied by either already installed modules or modules that exist within the packages being installed by the setScript. Insure to take into consideration that another package in the setScript could be removing a module that may be required.

[00177] 2) Check storage dependencies taking into account that a package within the setScript command could be removing a module and therefore freeing up storage.

[00178] 3) Wait for install time frame.

[00179] 4) Disable EGM according to disableType attribute.

[00180] 5) Initiate processing of packages according to the initiateType command.

[00181] 6) Process authorizations. There can be multiple authorizations required. This includes a local operator authorization as well as multiple host authorizations.

[00182] 7) Scripts may or may not contain command lists. If not command lists are included, then the package is installed based on the contents of the package. Command lists

will only exist for removing modules or executing specific commands on the EGM that is not related to installing or removing packages.

[00183] 8) Whenever a package is removed, its related file validation manifest must also be removed from the system

[00184] 9) Whenever a module is installed or removed from the EGM that cause a manifest to be modified, deleted or added, the system must be rebooted after the installation completes.

[00185] 10) Based on jurisdiction requirements and state specified in the setScript command, delete the downloaded package.

[00186] Installing and Updating Module Requirements

[00187] Whenever a module is installed or updated on a system that has sufficient storage to maintain a backup copy of the operating environment, the following steps are performed.

[00188] 1) Reset the partition access permissions to allow writing to the partitions.

[00189] 2) Copy the production environment into the backup environment. This may be done via a background task when an environment is activated and while the game is running.

[00190] 3) Apply the changes to the production environment.

[00191] 4) Insure that the boot.id file is set to boot the production environment and that a backup environment exists.

[00192] 5) Reboot the system according to jurisdictional requirements.

[00193] When processing the package, the package will either contain a tar file for updates to the system or an image of a partition or entire storage media. If there is an image file, a check is performed to insure that the image is the correct size for the media.

[00194] When installing new games, this is performed via a tar file. A check is made to insure that there is enough space to hold the new or updated game's files and manifest file. No backup will be made of an existing game on the system. If the game fails to run, we expect that it will have to be downloaded again from the server.

[00195] Installation Dependencies

[00196] Installation dependencies and pre-requisites are used interchangeably. Each may have a set of module, hardware and storage dependencies that must exist before the module can be installed. The dependency checking is performed as follows:

[00197] · Module Dependency – A module dependency is defined by it Module ID and Release Information.

[00198] · Hardware Dependency – The module dependency is defined by the Hardware ID and version number

[00199] · Storage Dependency – Defined by the storage type and the amount of free space required.

[00200] For Release Information and hardware version number, a test flag will define how to identify if a dependency is met. The dependency check flag will have the following values:

[00201] · 0 – No check is performed.

[00202] · 1 – The release number or version number must be equal to the one of the installed hardware or module.

[00203] · 2 – The release number version number must be greater than the installed one.

[00204] · 3 – The release or version must be greater than or equal to the installed one.

[00205] <u>setScript Command Structure</u>

[00206] This section describes the setScript command structure that will be passed into the download install logic.

[00207] Field Entry	**[00208]** Field Type	**[00209]** Description
[00210] setScript ID	**[00211]** string	**[00212]** Unique identifier for the setScript command
[00213] startTime	**[00214]** time_t	**[00215]** Specifies the start time frame of when the attached command can start processing
[00216] endTime	**[00217]** time_t	**[00218]** Specifies the end of the time window when the attached scripts can start processing

[00219] disableType	**[00220]** integer	**[00221]** Indicates the conditions under which the EGM is to be disabled to start processing the attached scripts
[00222] initiateType	**[00223]** integer	**[00224]** Indicates the events that need to happen in order to start processing the attached command list.
[00225] authorizeList	**[00226]** string array	**[00227]** A list of hosts that need to authorize the installation of the package.
[00228] packageList	**[00229]** string array	**[00230]** A list of package IDs to be processed by this script command.

[00231] startTime / endTime – This is a date and time stamp that defines the start of the time and end of a time window within which a setScript command can start processing. None of the packages within the package list starts processing before this date and time are reached. The endTime is the date and time stamp after which the setScript command cannot start. The start of processing depends upon the initiateType being satisfied and all the authorizations being met. If these are not met, then the processing of the setScript command is suspended until the time window is entered again. Once the first package has started processing, all other packages will be processed regardless of the time window.

[00232] disableType – This specifies how the EGM should be disabled. The EGM cannot be disabled until the time processing time window is entered. As soon as the disable conditions are met, the EGM will be disabled and wait for the authorizations to occur. If the authorizations do not occur within the processing time window, the setScript command will be discontinued and the EGM re-enabled. The setScript command is then placed back into the waiting to process queue.

[00233] initiateType – Specifies what actions need to take place in order to start the installation. This includes host authorizations, local operator authorization, etc. These events can occur before the EGM is disabled in the case of host authorizations. All initiation requirements must be satisfied during the process time window.

[00234] authorizeList – This is a list of host IDs who must authorize the setScript command to start processing. If the host specifies authorization not granted, then the processing of the setScript command will be terminated.

[00235] PackageList – This is a list of packages to be processed. The packages will be processed in the order that they are specified within the setScript command. Module dependencies within one package may be satisfied by module in another package within the package list.

[00236] When a package specifies that a module is to be deleted, then all the files within the Module manifest file will be deleted from the system along with the manifest file itself.

[00237] Download Package Description

[00238] This document is used to specify the detail design of download packages. Each download package is made of various sections. Each section starts with a section ID, the size of the section, and the contents of the section. The contents of the section is dependent upon the type of package section defined.

[00239] The specific package sections are:

[00240] · Package Header – This describes the overall package.

[00241] · Module Entry – A package may contain one or module entries.

[00242] · Dependency Entry – After each module. The hardware and software dependencies are defined for the module. There may none or multiple dependencies for each module.

[00243] · File Entry – Various types of file entries are defined. Each type has its own unique ID

[00244] All the sections after the package header are considered to be part of the package data. A SHA 1 hash value is calculated over the entire package data. This hash value does not include the header in its calculation.

[00245] Each package section contains an integer header. The first integer is the section ID, and the second is the size of the section itself. The following sections describe the information that is contained within each package section.

[00246] The contents of the package header can be seen in the following table:

[00247] Field	**[00248]**	**[00249]** Description
[00250] mcnt	**[00251]** integer	**[00252]** Number of modules contained within the package
[00253] ccnt	**[00254]** integer	**[00255]** Number of command defined in the package
[00256] compression	**[00257]** integer	**[00258]** Compression applied to the package data
[00259] psize	**[00260]** integer	**[00261]** Size of the package Data
[00262] hsize	**[00263]** integer	**[00264]** Size of the header data
[00265] type	**[00266]** integer	**[00267]** Type of Package
[00268] Time_stamp	**[00269]** Time_t	**[00270]** Time stamp of when the package was built
[00271] vString	**[00272]** char	**[00273]** Package Data Validation String (SHA1 or HMAC)
[00274] pkgId	**[00275]** char	**[00276]** Unique identifier of the package
[00277] release	**[00278]** char	**[00279]** Release

		information of package, (Version, build No, etc)
[00280] description	**[00281]** Char	**[00282]** Description of package contents
[00283] builderID	**[00284]** char	**[00285]** Identity of vender that created package

[00286] The package header is the first thing contained with a package and is a fixed length in size.

[00287] The package data compression types are:

[00288] Compression	**[00289]** Description
[00290] 1	**[00291]** Uncompressed data
[00292] 2	**[00293]** Compressed data using gzip protocol
[00294] 3	**[00295]** Compressed data using bzip2 protocol

[00296] The following package type are been identified:

[00297] Package Type ID	**[00298]** Description
[00299] 1	**[00300]** Download Package is download from a package distribution server
[00301] 2	**[00302]** Upload Package is created at the EGM and uploaded to a server.
[00303] 3	**[00304]** Command Package is downloaded from a server and only contains commands to be run at the EGM.

[00305] <u>Module Sections</u>

[00306] There can be multiple module sections within a package. The module section is identified by the package section ID number 2 and the size is the size of the module information as described in the table below.

[00307] Field	[00308] Type	[00309] Description
[00310] Type	[00311] integer	[00312] Type of Module
[00313] Action	[00314] Integer	[00315] Action to take on Module (Add, update, delete)
[00316] Mod Depend cnt.	[00317] Integer	[00318] No. of module dependencies for this module
[00319] Hard Depend cnt.	[00320] Integer	[00321] No. of hardware dependencies for this module
[00322] Strg Depend cnt	[00323] Integer	[00324] No. of defined storage dependencies
[00325] Time Stamp	[00326] integer	[00327] Time stamp associated with when module built.
[00328] Release Number	[00329] sring	[00330] Specific Release Major, Minor, Build and Version Nos.
[00331] Name	[00332] string	[00333] Null terminate string containing the name of the module.
[00334] Description	[00335] string	[00336] An optional null terminated string used to describe the module.

[00337] Each module is assigned a unique module name. The first three characters of the name are manufacturer unique and used to identify the manufacturer of the module. The size

of the name does not exceed 32 characters in length in one embodiment, but that is by way of example only.

[00338] The hardware, module, and storage dependency counts specify how many of each type of dependency is included in the package.

[00339] The type is used to identify the type of module. The following table identifiers the module type that are used:

[00340] Module Type ID	[00341] Description
[00342] os	[00343] EGM OS module. Linux and Game kernels, libraries and utilities
[00344] gm	[00345] EGM Game module.
[00346] fw	[00347] Firmware module
[00348] dt	[00349] Data module. Contains EGM unique data.
[00350] jr	[00351] Jurisdiction module
[00352] df	[00353] Definition Module – Used to define new modules within the EGM
[00354] cf	[00355] Configuration Module – Used to specify configuration information for the OS, EGM and Game.

[00356] <u>Package File Definitions</u>

[00357] Files entries within the package are associated with the preceding module identified in the package. That is, all files entries after a module definition is encountered until the next module or command definition are associated with the module they follow. In the following example, module 1 has file 1 and 2 associated with it and module 2 has files 3 and 4 associated with it.

[00358] Package Header
[00359] Module 1

[00360] File 1
[00361] File 2
[00362] Module 2
[00363] File 3
[00364] File 4

[00365] All file definitions use the same control structure.

[00366] Field	**[00367]** Type	**[00368]** Description
[00369] Type	**[00370]** Integer	**[00371]** Type file that follows
[00372] File size	**[00373]** integer	**[00374]** Size of the file
[00375] File name	**[00376]** string	**[00377]** File name including fully qualified path.
[00378] Destination	**[00379]** string	**[00380]** Fully qualified path where file is to be stored. Used for image files.

[00381] The following describes the type of files that can be included:

[00382] The manifest file section data area comprises the following 3 fields:

[00383] Field	**[00384]** type	**[00385]** Description
[00386] Type	**[00387]** integer	**[00388]** Type of manifest, OS (1), Game (2)
[00389] Manifest Name	**[00390]** string	**[00391]** Manifest file name (32 Characters Max)

[00392] File data	[00393] binary	[00394] The manifest file contents

[00395] The type field is used to determine which manifest sub-directory to copy the manifest file into. Manifests will be copied into a directory called manifests and either into the OS1 or games subdirectory.

[00396] Tar File – (Package ID Type: 4)

[00397] Updates to installed modules as well as game installations will be in the form of a tar file.

[00398] Partition Image File – (Package ID Type: 5)

[00399] The partition image section contains a complete image of a particular partition on the EGM. Following the Section header data are the identification of the partition to be replaced by the image, followed by the image itself. The EGM will assign a name to the image file for processing at the EGM. The partition identification is a null terminated string. The partition that contains the manifest validation files will always be partition one on any device. This partition can not be updated with the use of a partition image.

[00400] Device Image File – (Package ID Type: 6)

[00401] The device image file section contains the complete image of a device on the EGM such as an entire compact flash. When the device image is specified, it is assumed that the manifest files are include within the image file itself and are therefore not included in a package manifest section. Following the section header, is a null terminated string that contains the device the image is meant for, such as /dev/had, followed by the image itself. The EGM will assign a name to the image file locally.

[00402] Dependency Sections

[00403] Hardware Dependencies (Package ID Type: 7)

[00404] The hardware dependency section identifies what hardware must exist on the system before the previously defined module can be installed. There can be any number of these associated with each module in the package. The hardware dependency section is comprised of the hardware ID as known on the EGM, a version number and the comparison type to be performed (equal, greater than, or greater than or equal).

[00405] Module Dependencies (Package ID Type: 8)

[00406] The module dependency specifies which modules must exist on the EGM before the previously defined module can be installed. There can be any number of these associated with each module in the package. The module dependency consist of the module id, the version number for the module and the comparison identifier that specifies the type of comparison to be performed against the version number (equal to, greater then, or greater than or equal to).

[00407] Storage Dependency (Package ID Type: 9)

[00408] The storage dependency defines how much storage must exist on the EGM. The fields in the storage record specify the type of storage, RAM, ROM, disk, etc and how mush must be there or how much free space must exist depending on the type of storage specified.

[00409] Package Record Identifiers

[00410] This describes the currently assigned id values for download package sections.

[00411] Package Section Type	**[00412]** ID	**[00413]** Description
[00414] Package Header	**[00415]** 1	**[00416]** Information unique to the package.
[00417] Module Definition	**[00418]** 2	**[00419]** Defines modules included in package
[00420] Manifest File	**[00421]** 3	**[00422]** Manifest File Associated with previous module definition.
[00423] Tar File	**[00424]** 4	**[00425]** Tar file associated with previous Module definition
[00426] Partition Image	**[00427]** 5	**[00428]** Partition Image file associated with previous Module definition.

[00429] Device Image file	**[00430]** 6	**[00431]** Device Image file associated with previous Module definition.
[00432] Hardware Pre-requisite	**[00433]** 7	**[00434]** Hardware pre-requisite/dependency associated with previous module definition.
[00435] Module Pre-requisite	**[00436]** 8	**[00437]** Module Pre-requisite/dependency associated with previous module definition.
[00438] Storage Dependency	**[00439]** 9	**[00440]** Storage Pre-requisite dependency associated with previous module definition.

CLAIMS

What is claimed:

1. A system for providing software to one or more gaming machines comprising:

a software distribution point for generating a download package and sending the package as a plurality of packets to the one or more gaming machines as part of a multicast;

at each gaming machine receiving the packets and determining if all packets have been received;

requesting dropped packets from the software distribution point:

resending all dropped packets in a multicast.

2. The system of claim 1 wherein the multicast is done as a background process.

3. The system of claim 1 wherein a download control process on a gaming machine communicates with a system management point to control the download of the package.

4. The system of claim 3 further including a download receiver, download driver, and package installer on the gaming machine.

5. The system of claim 4 further including validating the package before installation.

6. The system of claim 5 further including checking for software prerequisites before installing the package.

7. The system of claim 6 further including downloading software prerequisites when needed before installing the package.

8. The system of claim 7 further including checking for hardware prerequisites before installing the package.

9. The system of claim 1 further including dividing memory on a gaming machine into partitions.

10. The system of claim 9 wherein the partitions comprise a manifest partition, a games partition and a download partition.

11. The system of claim 8 further including disabling the game machine when installation conditions have been met.

12. A system of initializing a gaming machine comprising:

a storage media having boot-strap instruction on the gaming machine;

communications means using the boot-strap instructions to initialize communications with a server upon initial start-up of the gaming machine and requesting a software package from the server;

the gaming machine receiving the software package and installing the software package on the gaming machine using the boot-strap instructions.

13. The system of claim 12 wherein the boot-strap instructions include a default address of a server for communication.

14. The system of claim 13 wherein the server is an SMP network server.

15. The system of claim 12 wherein the boot-strap instructions request a default boot-strap package from the server.

16. The system of claim 15 wherein the server is an SDP.

17. A method for providing software to one or more gaming machines comprising:

generating a download package at a software distribution point;

sending the package as a plurality of packets to the one or more gaming machines as part of a multicast;

receiving the packets at each gaming machine;

determining if all packets have been received;

requesting dropped packets from the software distribution point:

resending all dropped packets in a multicast.

18. The method of claim 17 wherein the multicast is done as a background process.

19. The method of claim 17 wherein a download control process on a gaming machine communicates with a system management point to control the download of the package.

20. The method of claim 19 further including a download receiver, download driver, and package installer on the gaming machine.

21. The method of claim 20 further including validating the package before installation.

22. The method of claim 21 further including checking for software prerequisites before installing the package.

23. The method of claim 22 further including downloading software prerequisites when needed before installing the package.

24. The method of claim 23 further including checking for hardware prerequisites before installing the package.

25. The method of claim 17 further including dividing memory on a gaming machine into partitions.

26. The method of claim 25 wherein the partitions comprise a manifest partition, a games partition and a download partition.

27. The method of claim 24 further including disabling the game machine when installation conditions have been met.

28. A method of initializing a gaming machine comprising:
providing boot-strap instructions on a storage media on the gaming machine;
using the boot-strap instructions to initialize communications with a server upon initial start-up of the gaming machine;
requesting a software package from the server;
receiving the software package and installing the software package on the gaming machine using the boot-strap instructions.

29. The method of claim 28 wherein the boot-strap instructions include a default address of a server for communication.

30. The method of claim 29 wherein the server is an SMP network server.

31. The method of claim 28 wherein the boot-strap instructions request a default boot-strap package from the server.

32. The method of claim 31 wherein the server is an SDP.

FIGURE 1

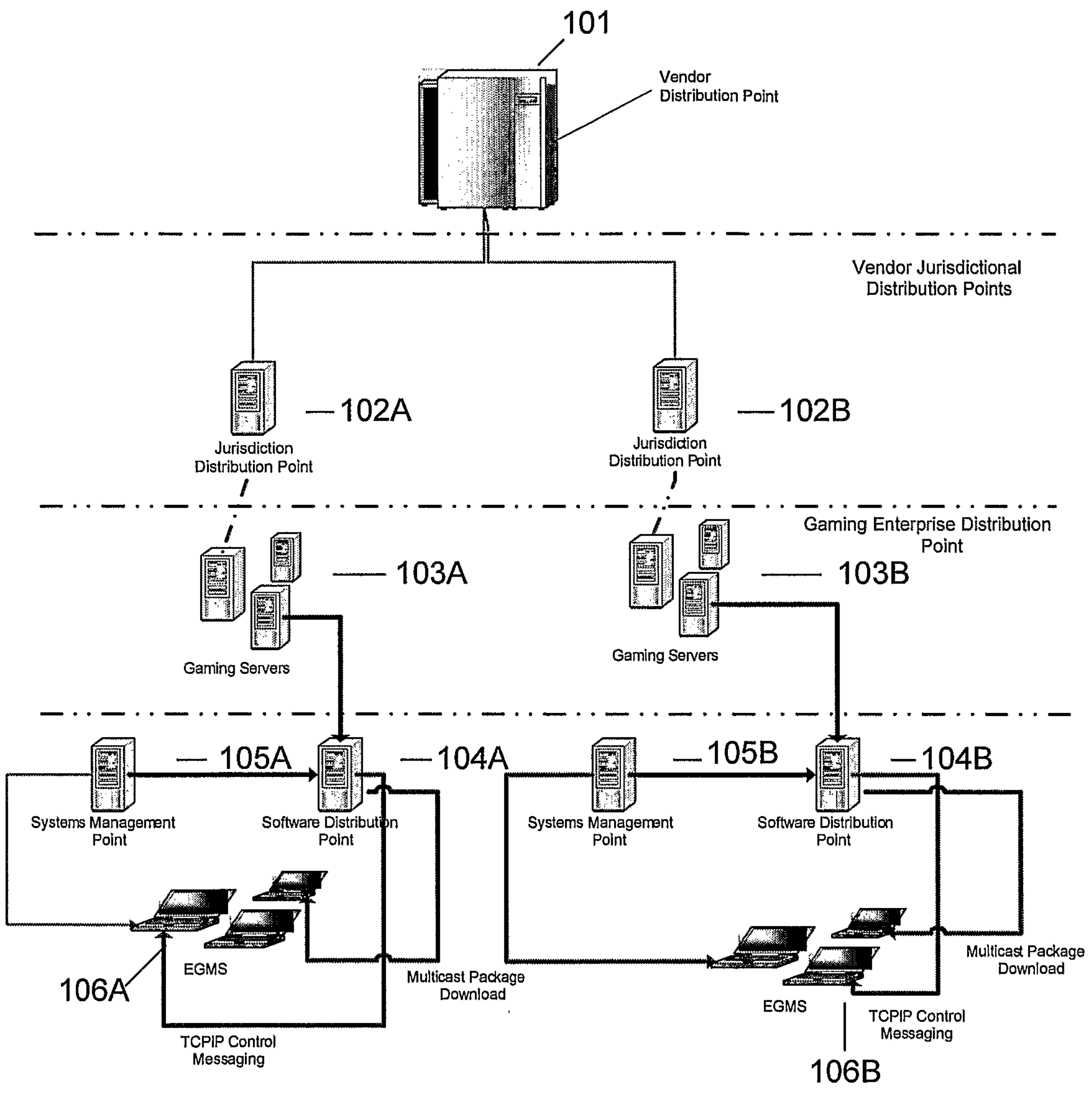

101
Vendor Distribution Point
Vendor Jurisdictional Distribution Points
102A
Jurisdiction Distribution Point
102B
Jurisdiction Distribution Point
Gaming Enterprise Distribution Point
103A
Gaming Servers
103B
Gaming Servers
105A
104A
Systems Management Point
Software Distribution Point
105B
104B
Systems Management Point
Software Distribution Point
EGMS
106A
Multicast Package Download
TCPIP Control Messaging
Multicast Package Download
EGMS
TCPIP Control Messaging
106B

FIGURE 2

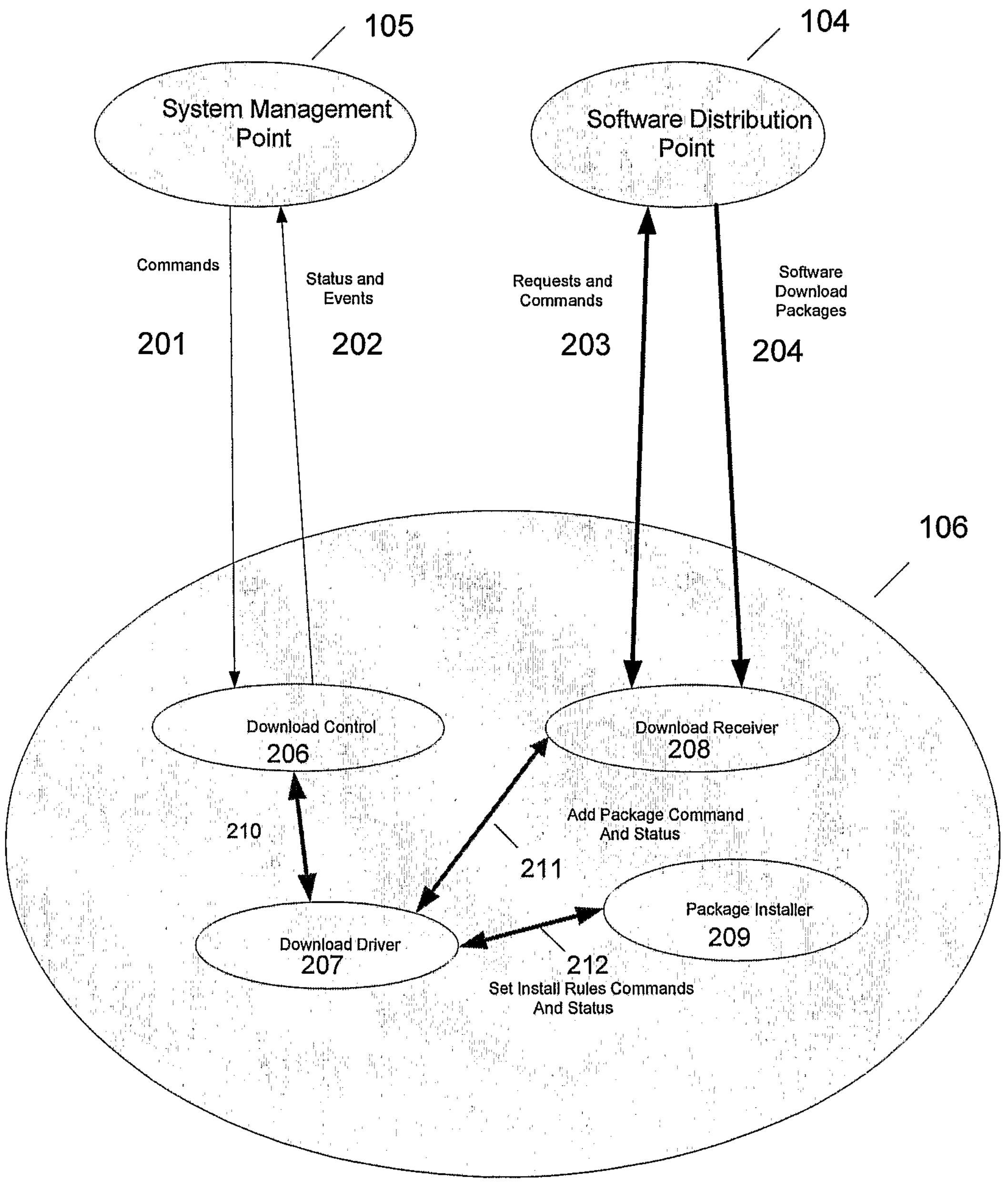

105
System Management Point
104
Software Distribution Point
Commands
201
Status and Events
202
Requests and Commands
203
Software Download Packages
204
106
Download Control
206
Download Receiver
208
210
Add Package Command And Status
211
Package Installer
209
Download Driver
207
212
Set Install Rules Commands And Status

FIGURE 3

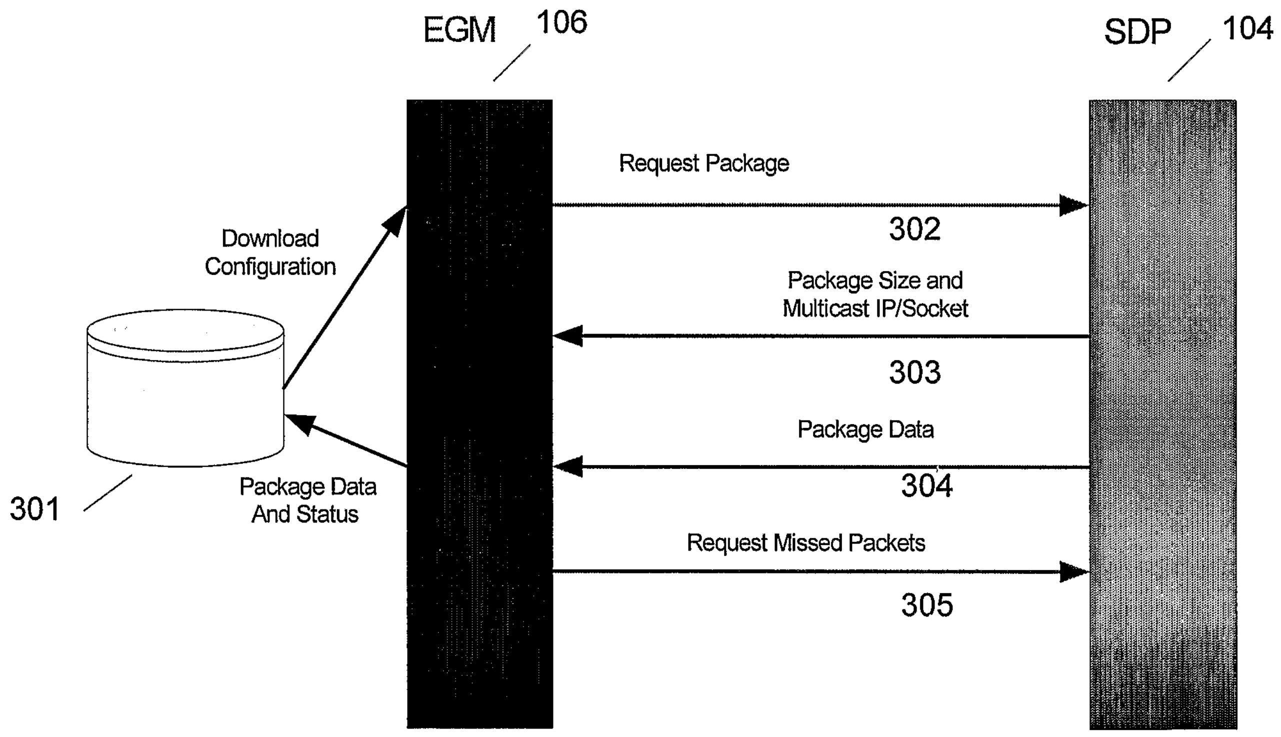

EGM
106
SDP
104
Request Package
302
Download Configuration
Package Size and Multicast IP/Socket
303
Package Data
304
301
Package Data And Status
Request Missed Packets
305

FIGURE 4

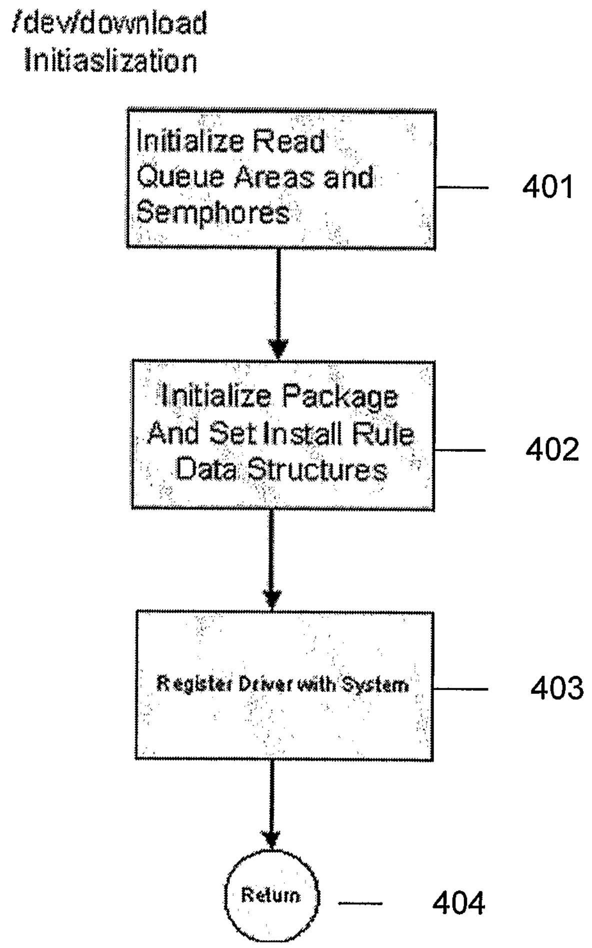

/dev/download
Initiaslization
Initialize Read Queue Areas and Semphores
401
Initialize Package And Set Install Rule Data Structures
402
Register Driver with System
403
Return
404

FIGURE 5

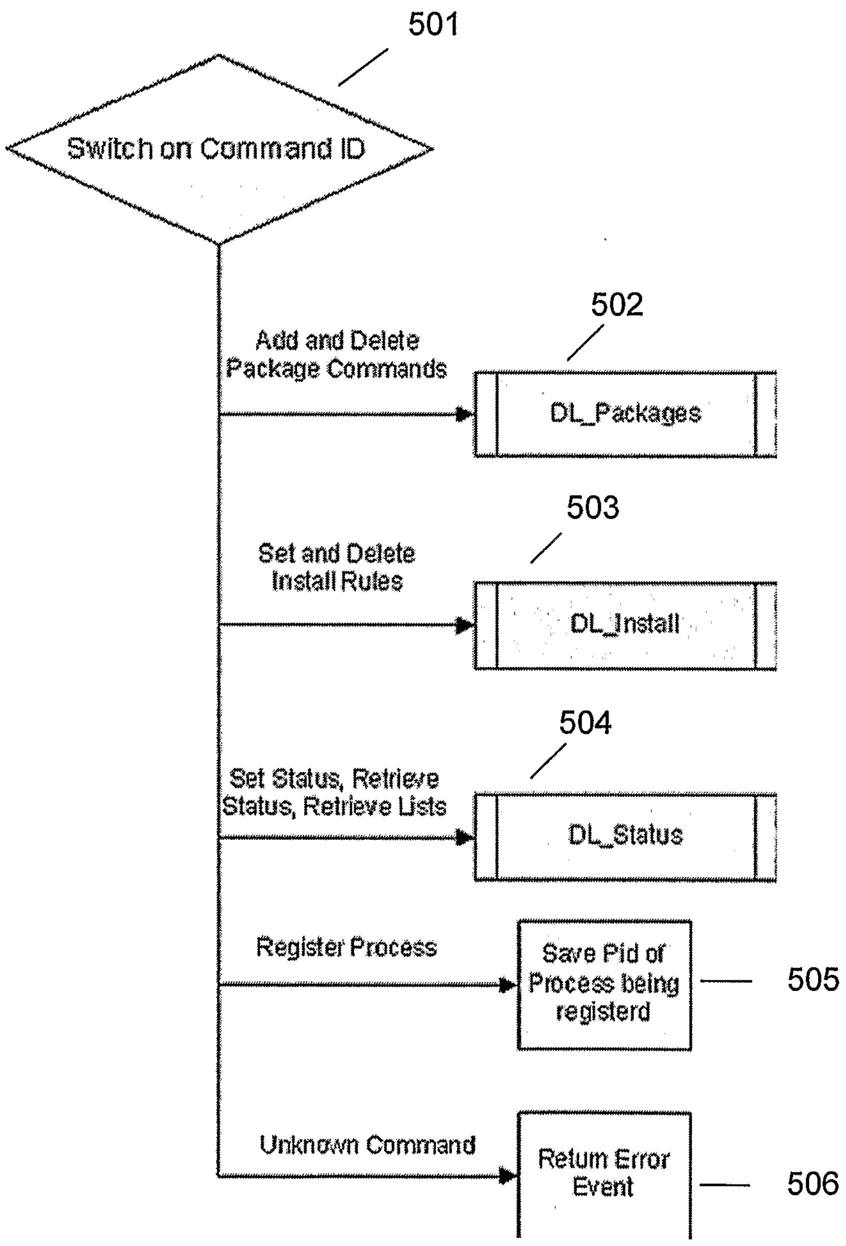
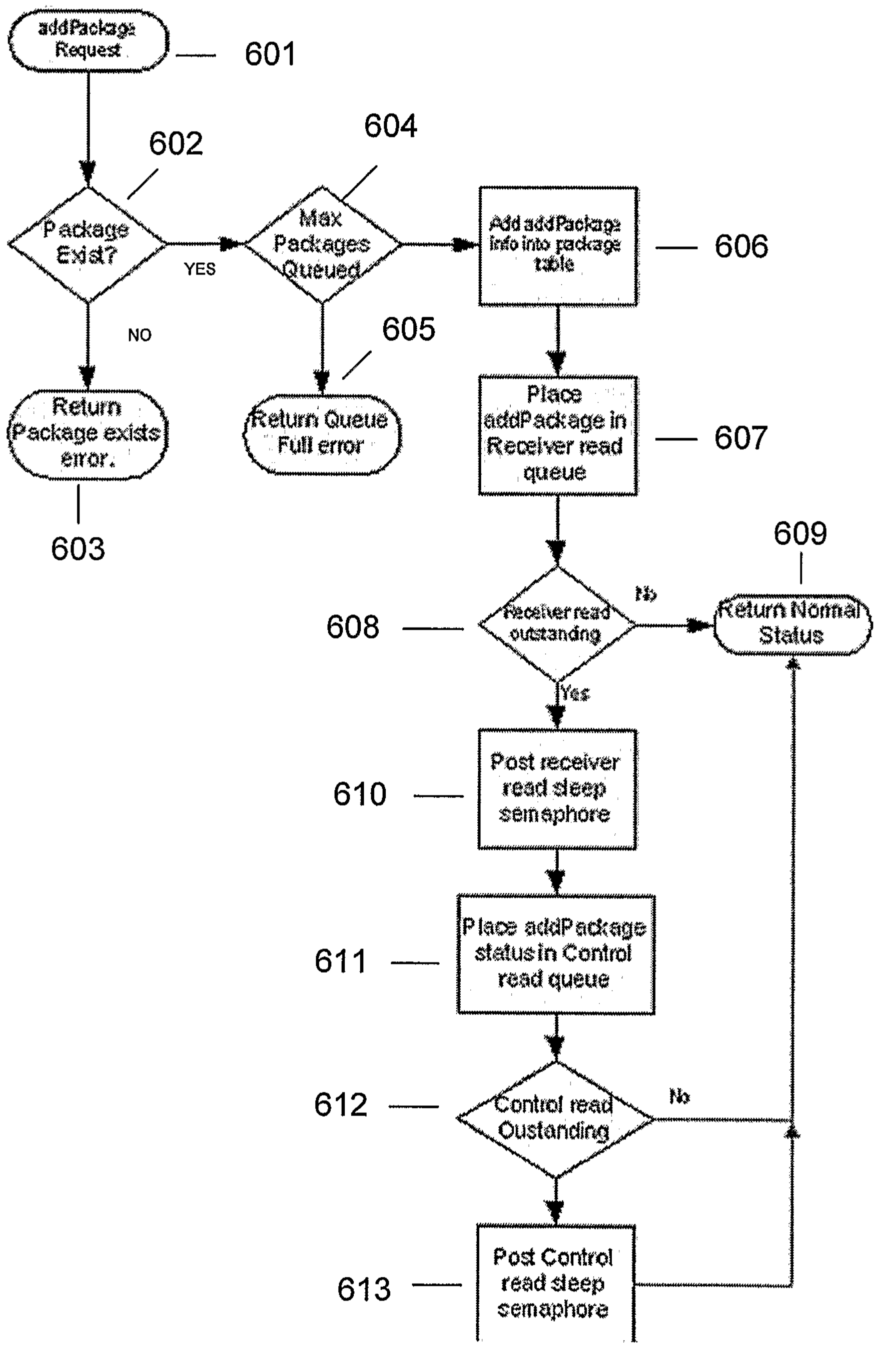

501
Switch on Command ID
Add and Delete Package Commands
502
DL_Packages
Set and Delete Install Rules
503
DL_Install
Set Status, Retrieve Status, Retrieve Lists
504
DL_Status
Register Process
Save Pid of Process being registerd
505
Unknown Command
Return Error Event
506

FIGURE 6



addPackage Request
601
Package Exist?
602
YES
NO
Max Packages Queued
604
Return Package exists error.
603
Return Queue Full error
605
Add addPackage info into package table
606
Place addPackage in Receiver read queue
607
Receiver read outstanding
608
No
Yes
Return Normal Status
609
Post receiver read sleep semaphore
610
Place addPackage status in Control read queue
611
Control read Oustanding
612
No
Post Control read sleep semaphore
613

FIGURE 7

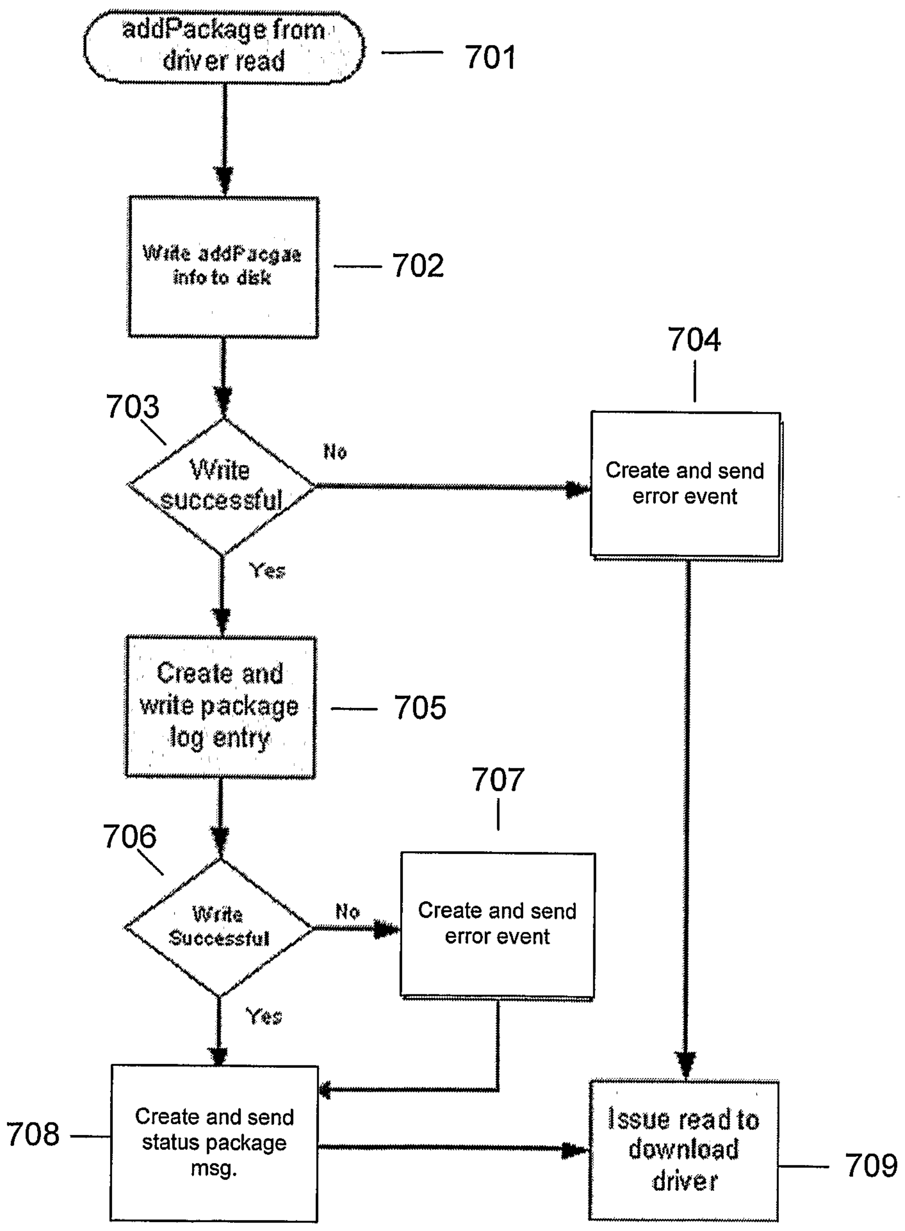

addPackage from driver read
701
Write addPacgae info to disk
702
703
Write successful
No
704
Create and send error event
Yes
Create and write package log entry
705
706
Write Successful
No
707
Create and send error event
Yes
708
Create and send status package msg.
Issue read to download driver
709

FIGURE 8

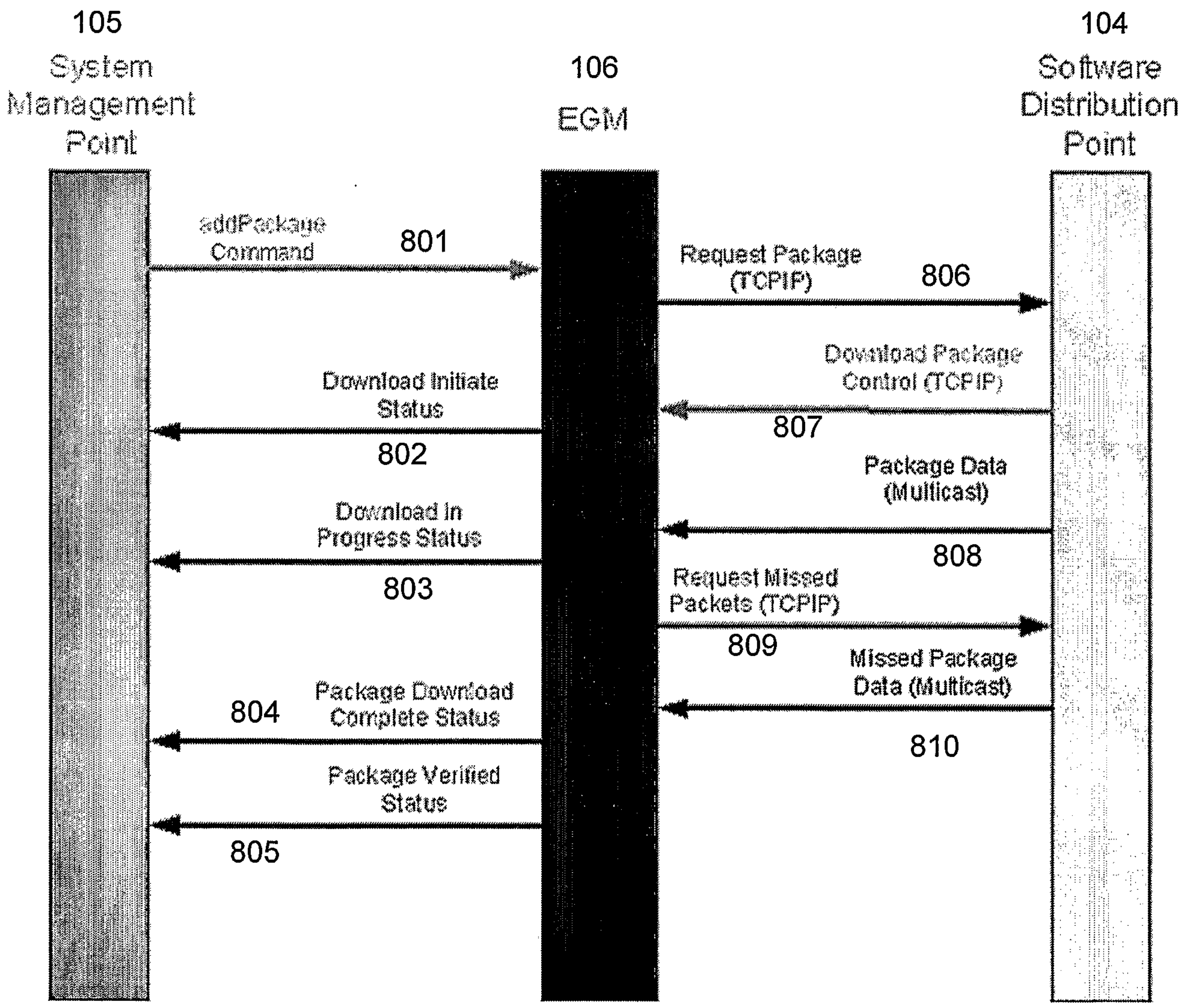

105
System Management Point
106
EGM
104
Software Distribution Point
addPackage Command
801
Request Package (TCPIP)
806
Download Package Control (TCPIP)
807
Download Initiate Status
802
Package Data (Multicast)
808
Download In Progress Status
803
Request Missed Packets (TCPIP)
809
Missed Package Data (Multicast)
810
Package Download Complete Status
804
Package Verified Status
805

FIGURE 9

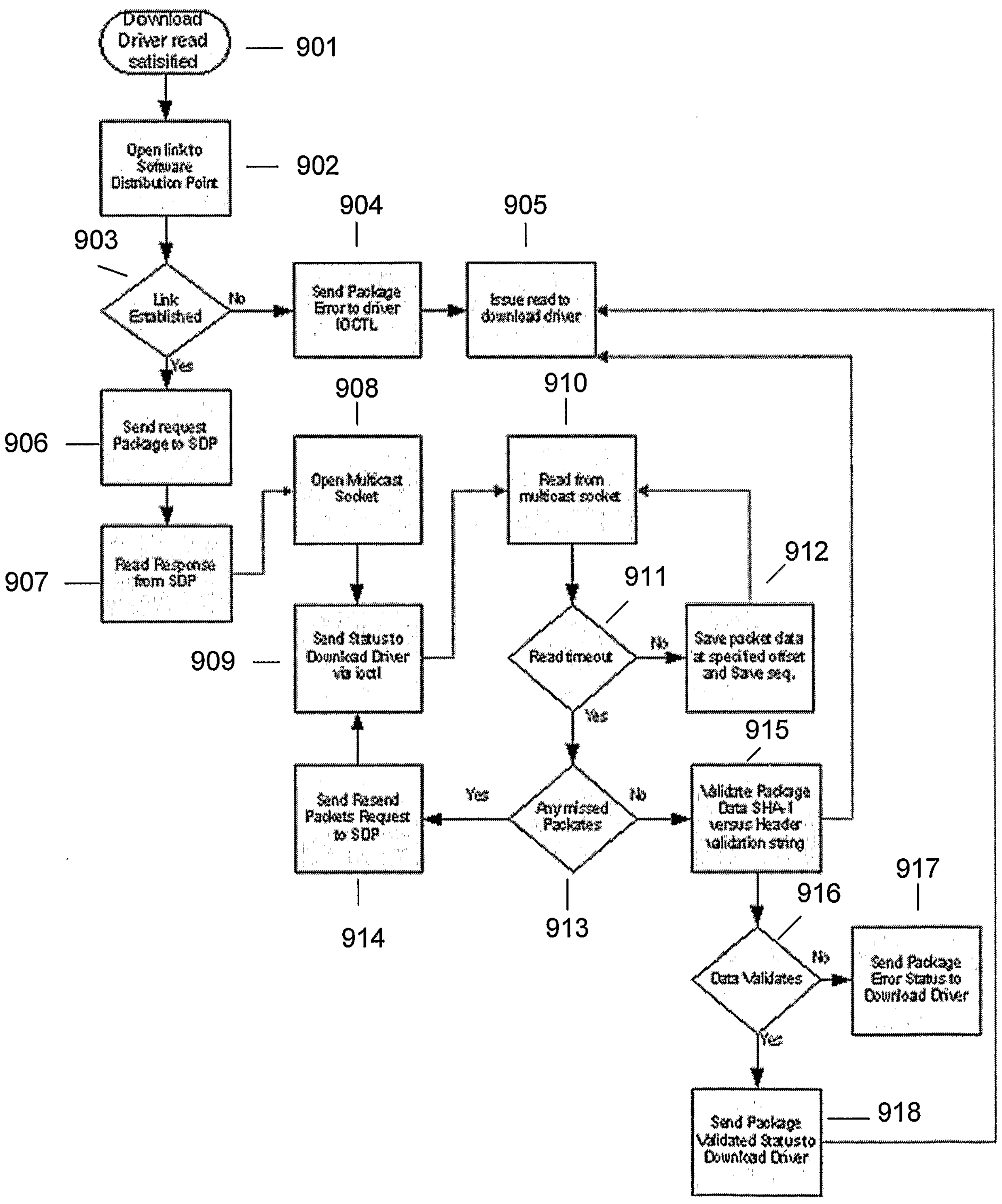

Download Driver read satisified
901
Open link to Software Distribution Point
902
903
Link Established
No
904
Send Package Error to driver IOCTL
905
Issue read to download driver
Yes
906
Send request Package to SDP
907
Read Response from SDP
908
Open Multicast Socket
909
Send Status to Download Driver via ioctl
910
Read from multicast socket
911
Read timeout
No
912
Save packet data at specified offset and Save seq.
Yes
913
Any missed Packates
Yes
914
Send Resend Packets Request to SDP
No
915
Validate Package Data SHA-1 versus Header validation string
916
Data Validates
No
917
Send Package Error Status to Download Driver
Yes
918
Send Package Validated Status to Download Driver

FIGURE 10

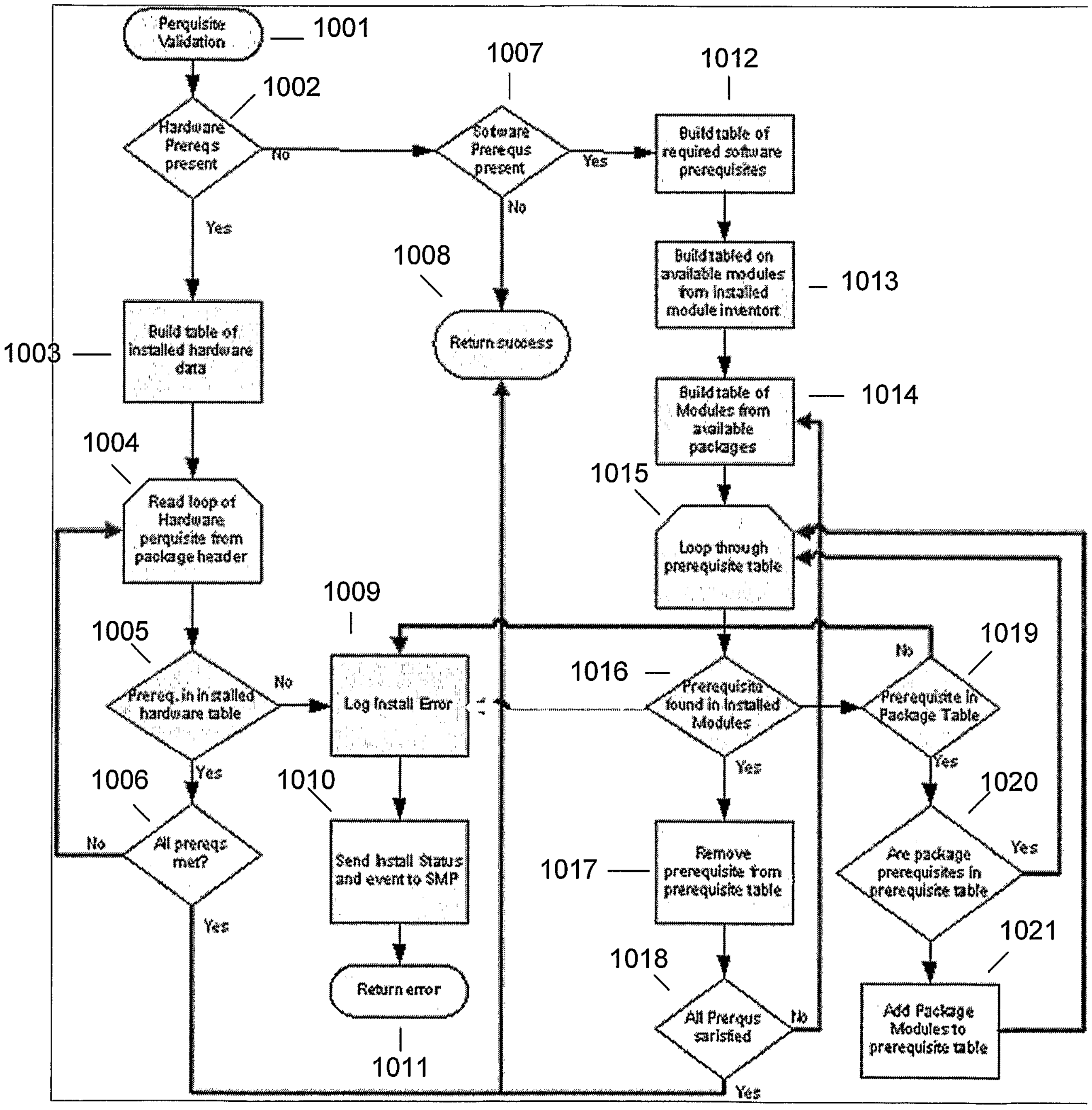

Perquisite Validation
1001
Hardware Prereqs present
1002
No
Yes
Build table of installed hardware data
1003
Read loop of Hardware perquisite from package header
1004
Prereq. in installed hardware table
1005
No
Yes
All prereqs met?
1006
No
Yes
Software Prerequs present
1007
Yes
No
Return success
1008
Log Install Error
1009
Send Install Status and event to SMP
1010
Return error
1011
Build table of required software prerequisites
1012
Build tabled on available modules from installed module inventort
1013
Build table of Modules from available packages
1014
Loop through prerequisite table
1015
Prerequisite found in installed Modules
1016
Yes
Remove prerequisite from prerequisite table
1017
All Prerqus satisfied
1018
No
Yes
Prerequisite in Package Table
1019
No
Yes
Are package prerequisites in prerequisite table
1020
Yes
Add Package Modules to prerequisite table
1021

FIGURE 11

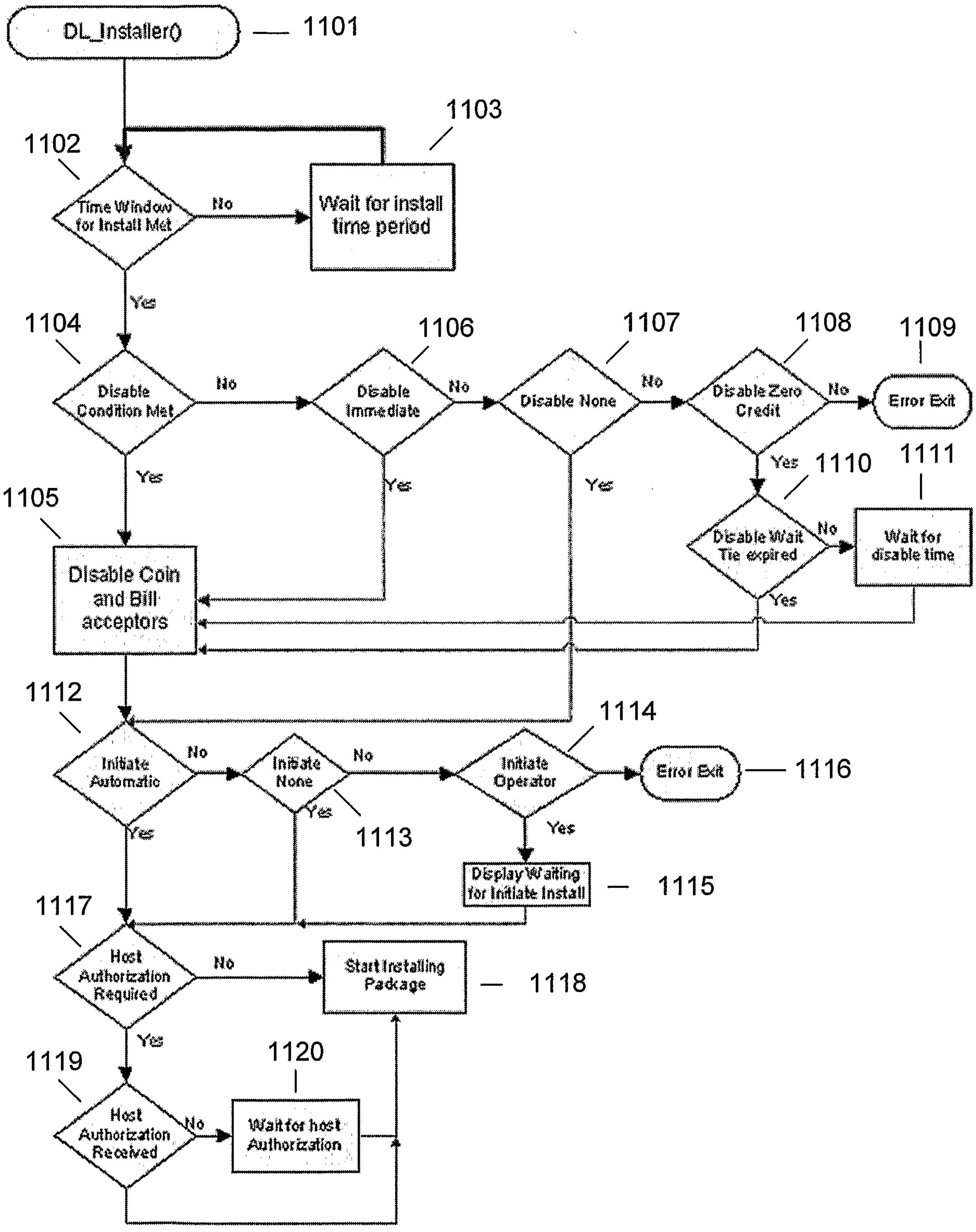
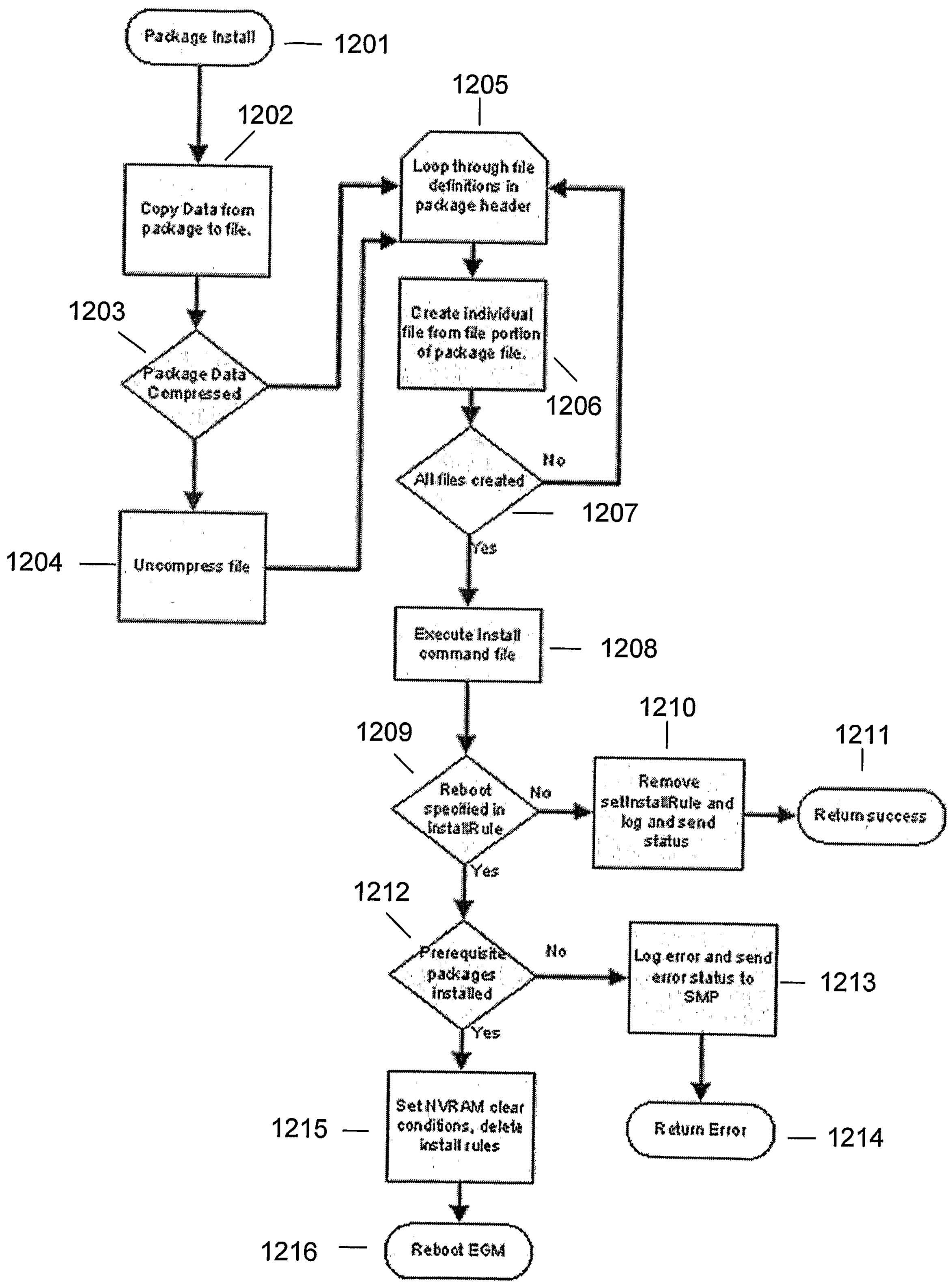

DL_Installer()
1101
1102
Time Window for Install Met
No
Wait for install time period
1103
Yes
1104
Disable Condition Met
No
1106
Disable Immediate
No
1107
Disable None
No
1108
Disable Zero Credit
No
Error Exit
1109
Yes
Yes
Yes
Yes
1110
1111
1105
Disable Coin and Bill acceptors
Disable Wait Tie expired
No
Wait for disable time
Yes
1112
Initiate Automatic
No
Initiate None
No
1114
Initiate Operator
Error Exit
1116
Yes
Yes
1113
Yes
Display Waiting for Initiate Install
1115
1117
Host Authorization Required
No
Start Installing Package
1118
Yes
1119
Host Authorization Received
No
1120
Wait for host Authorization

FIGURE 12


Package Install
1201
Copy Data from package to file.
1202
Package Data Compressed
1203
Uncompress file
1204
Loop through file definitions in package header
1205
Create individual file from file portion of package file.
1206
All files created
No
Yes
1207
Execute Install command file
1208
Reboot specified in InstallRule
No
Yes
1209
Remove setInstallRule and log and send status
1210
Return success
1211
Prerequisite packages installed
No
Yes
1212
Log error and send error status to SMP
1213
Return Error
1214
Set NVRAM clear conditions, delete install rules
1215
Reboot EGM
1216