IN-SERVICE SOFTWARE UPGRADE UTILIZING METADATA-DRIVEN STATE TRANSLATION

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims the benefit of U.S. Provisional Patent Application Number 60/995,882, entitled "IN-SERVICE SOFTWARE UPGRADE UTILIZING METADATA-DRIVEN STATE TRANSLATION" filed on September 27, 2007.

BACKGROUND

Field

[0002] Embodiments of the invention relate to the field of computer networking; and more specifically, to upgrading network elements utilizing metadata state translation.

Background

[0003] A network element in a service provider's network typically handles high volumes of data traffic from users accessing several different services and/or communicating with other users. For example, a network element can handle services for up to tens of thousands of users. An interruption in the operation of the network element can cause a disruption of service to these tens of thousand of users.

[0004] Figure 1 illustrates one embodiment of a service provider network 100 with a network element forwarding traffic between end stations and content servers through a network. While in one embodiment, network element is a router, in alternate embodiment, network element can be other networking equipment known in the art (switch, hub, firewall, etc.). Furthermore, while in one embodiment network element forwards traffic, in alternate embodiment,

networking can perform the same and/or different traffic processing (switch traffic, shape traffic, smooth traffic, apply access controls, apply firewall policies, etc.) In Figure 1, network 100 comprises end stations 102A-C, network element 104, core network 106, and content servers 108A-B. End stations 102A-C couple to network element 104 and network element couples to content servers 108A-B through core network 106. While in one embodiment, end station 102A-C are home personal computers, in alternate embodiments, end stations can be the same or different type of machine (e.g., business computer, personal digital assistant, cell phone, game console, handheld game system, laptop computer, etc.). End stations can couple to network element through any one of the means know in the art (e.g., Ethernet, wireless, digital subscriber line (DSL), cable modem, fiber, etc.). Network element 104 provides an entry point into core network 106 by forwarding data traffic from end stations 102A-C to content servers 108A-B, from content servers 108A-B to end stations 102A-C, and data traffic going between end stations 102A-C. While in one embodiment, network element 104 is an edge network element that forwards traffic between the edge network servicing end stations 102A-C and core network 106, in alternate embodiments, network element 104 can be a network element or switch positioned differently in the service provider's edge and/or core networks.

[0005] Core network 106 is the backbone network of the service provider that typically has high capacity to handle that high volume of traffic traveling through network 100. Content servers 108A-B serve content and/or control information for services offered to end stations 102A-C.

[0006] As network element 104 handles the data for this large number of users, network element 104 builds up a state that controls the handling of the data. While in one embodiment the state built up by network element 104 is a traffic forwarding table, in alternate embodiments, the state built up has the same and/or different information (configuration data, uses session information, firewall information, access control lists, quality of service information, statistics, etc.).

This state is typically run-time information that does not survive a reboot of network element 104.

[0007] Periodically, a network element receives a software upgrade to its services. Typically, a software upgrade requires a reboot of the network element so that software upgrade can take effect. However, a reboot disrupts the service and wipes out the built up state, because the state does not survive a reboot. Even thought a reboot of a network element can occur quickly, the rebuilding of the state typically takes longer. Rebuilding of the state involves reconnecting subscribers, rebuilding forwarding tables, subscriber session information, etc.

[0008] An improved software upgrade method, termed a "hitless" upgrade, is used when the network element has one or more redundant controllers. For example, a network element, such as network element 104, that has two or more controller cards can utilize a hitless upgrade. A hitless upgrade involves first installing and initializing the new software image of a backup or standby controller, synchronizing the network element state to the backup controller, switching control to the backup controller and then driving the software upgrade and state restoration to the other network element components. In this algorithm, the backup controller becomes the active controller for the network element and the former active controller is the backup controller.

[0009] Figure 2 is a block diagram of a network element 200 that includes redundant controller card. In Figure 2, backplane 206 couples to line cards 202A-N and controller cards 204A-B. While in one embodiment, controller cards 204A-B control the processing of the traffic by line cards 202A-N, in alternate embodiments, controller cards 204A-B perform the same and/or different functions (upgrade software, handle operator requests, collect statistics, etc.) Line cards 202A-N process and forward traffic according to the policies received from controller cards 204A-B.

[0010] Figure 3 is a block diagram of an active control card passing active state information to a backup control card. While in one embodiment, active state information is the state used by network element 104 as described in reference with Figure 1, in alternate embodiment, active state information is the same and/or different information used to operate network element 104. In Figure 3, active control card 302A sends active state information 304 to backup control card 302B. The hitless upgrade algorithm uses the active state information to perform the upgrade of the network element without a disruption of traffic processing.

[0011] A drawback of the hitless upgrade is that the upgrade needs to account for the differences between the old and the new images in the expected format and semantic content of the state. The hitless upgrade algorithm requires that the new software image contain explicit knowledge of the precise format and the content of the state data synchronized form the controller running the older image or that the data be transported in a version-independent format such as tag-length-value (TLV).

[0012] BRIEF SUMMARY

[0013] A method and apparatus for a metadata-driven upgrade of a network element is described. A network element comprises an active and backup controller card and receives new software to be installed on that network element. In addition, the network element installs the software on the backup controller card and synchronizes an active state of the network element on the backup controller card. Furthermore, the network element translates the active state to a new active state associated with the new software using a set of metadata associated with the new software.

BRIEF DESCRIPTION OF THE DRAWINGS

[0014] Embodiments of the invention may be best understood by referring to the following description and accompanying drawings which illustrate such

embodiments. The numbering scheme for the Figures included herein is such that the leading number for a given element in a Figure is associated with the number of the Figure. However, element numbers are the same for those elements that are the same across different Figures. In the drawings:

[0015] Figure 1 (Prior Art) illustrates one embodiment of network element handling traffic between end stations content servers through a network.

[0016] Figure 2 (Prior Art) is a block diagram of a network element with redundant controller cards.

[0017] Figure 3 (Prior Art) is a block diagram of an active control card passing active state information to a backup control card.

[0018] Figure 4 is a block diagram illustrating peer detection and processing system according to one embodiment of the system.

[0019] Figure 5 is an exemplary flow diagram for performing an in-service upgrade using metadata state translation according to one embodiment of the system.

[0020] Figure 6 is an exemplary flow diagram for creating the metadata according to one embodiment of the system.

[0021] Figure 7 is a block diagram illustrating a control card according to one embodiment of the invention.

DETAILED DESCRIPTION

[0022] In the following description, numerous specific details such as network element, metadata, data definition language, active state information, packet, processor card, line card, controller card, and interrelationships of system components are set forth in order to provide a more thorough understanding of the invention. It will be appreciated, however, by one skilled in the art that the invention may be practiced without such specific details. In other instances, control structures, gate level circuits and full software instruction sequences have

not been shown in detail in order not to obscure the invention. Those of ordinary skill in the art, with the included descriptions, will be able to implement appropriate functionality without undue experimentation.

[0023] References in the specification to "one embodiment", "an embodiment", "an example embodiment", etc., indicate that the embodiment described may include a particular feature, structure, or characteristic, but every embodiment may not necessarily include the particular feature, structure, or characteristic. Moreover, such phrases are not necessarily referring to the same embodiment. Further, when a particular feature, structure, or characteristic is described in connection with an embodiment, it is submitted that it is within the knowledge of one skilled in the art to affect such feature, structure, or characteristic in connection with other embodiments whether or not explicitly described.

[0024] In the following description and claims, the term "coupled," along with its derivatives, is used. "Coupled" may mean that two or more elements are in direct physical or electrical contact. However, "coupled" may also mean that two or more elements are not in direct contact with each other, but yet still co-operate or interact with each other. Exemplary embodiments of the invention will now be described with reference to Figures 4-7. In particular, the operations of the flow diagrams in Figures 5 and 6 will be described with reference to the exemplary embodiments of Figures 4 and 7. However, it should be understood that the operations of these flow diagrams can be performed by embodiments of the invention other than those discussed with reference to Figures 4 and 7 and that the embodiments discussed with reference to Figures 4 and 7 can perform operations different than those discussed with reference to these flow diagrams.

[0025] A method and apparatus for upgrading a network element using metadata is described. According to one embodiment of the invention, the network element uses metadata that describes the fields and structures of the active state maintained by the network element. The active state is information maintained by the network element on the state of traffic that is, was or will be processed by the

network element. Furthermore, the active state is information that does not survive a reboot of the network element. The network element translates the current active state using the metadata to an active state that supports new software associated with the upgrade.

[0026] Figure 4 is a block diagram illustrating network element software upgrade system according to one embodiment of the system. While in one embodiment, the software is a new or upgraded network element operating system, in alternate embodiments, the software is one or more new and/or upgraded services used by the network element. In Figure 4, active control card 418A sends active state information 416 to backup control card 418B. In one embodiment, active control card 418A and backup control 418B card run different versions of software. While in one embodiment, the software running on backup control card 418B is an upgraded version of software running on the active control card 418A, in alternative embodiments, the software versions running on control cards 418A-B can be the same and/or different software. As described in Figure 1, active state information 416 can comprise dynamic state built up during runtime. In one embodiment, active state information is the state built up as described above. Backup control card 418B comprises translation module 420 that translates active state information 416 from the software image running on active control card 418A to a form for use with the upgraded software image running on backup control card 418B. While in one embodiment, translation module 420 uses metadata 404 for the active state information 416 translation, in alternate embodiments, translation modules used the same and/or different information for the active state translation.

[0027] In addition, in Figure 4, metadata 404 is generated from data definition language (DDL) data 402. While in one embodiment metadata 404 is defined in extensible markup language (XML), in alternative embodiments, metadata 404 is defined in alternative format (plain text, a proprietary format, etc.). Furthermore, in one embodiment, metadata 404 is extensible and version independent. In one

embodiment, metadata 404 describes the structure and fields of active state information 416. In this embodiment, metadata 404 describes the constituent fields of each active state information 416 data structure, including the name, type, size, and offset in that data structure. While in one embodiment, metadata 404 supports data types such as strings, integers, real numbers, Booleans, in alternative embodiments, metadata 404 supports the same and/or other data types (enumerations, constants, bitmasks, type definitions, etc.). Furthermore, data structures can be defined in metadata 404 using such data types.

[0028] While in one embodiment, the DDL resembles the programming language C's data definitions with additional fields and features for defining information useful for data conversion and translation, in alternate embodiments, the DDL can be another language that supports defining data types and structures (another programming language, XML, plain text, etc.).

[0029] In one embodiment, DDL data 402 contains the data that describes the structure and fields of active state information 416 in the DDL format. In one embodiment, DDL data is processed into source code headers 406. While in one embodiment source code headers 406 are C language headers, in alternative embodiments, source code headers 406 can be in a different language header (C++, Pascal, etc.). In one embodiment, compiler 410 compiles source code 408 and source code headers 406 into network element operating system 412. While in one embodiment, software 412 is the network element operating system that runs the network element, including the software that controls the traffic forwarding, processing, shaping, etc. functions of the network element, in another embodiment, software 412 is one or more services used by the network element. As described above, software 412 is sent to backup control card 418B to upgrade the network element.

[0030] Figure 5 is an exemplary flow diagram of a method 500 for performing an in service upgrade using metadata state translation according to one embodiment of the system. In Figure 5, at block 502, method 500 receives the new software

image. In one embodiment, the network element receives the new software image.

[0031] At block 504, method 500 installs the software on the active controller card. Method 500 installs the software on the backup controller card at block 506.

[0032] At block 508, method 500 receives the new software metadata from the active controller card. In one embodiment, the received metadata is the structure and fields of active state information 416 associated with the new software and are generated from DDL data 402 as discussed in reference to Figure 4 above.

[0033] At block 510, method 500 synchronizes the backup controller active state information to the active controller active state information. In one embodiment, method 500 sends active state information 416 from the active controller card 418A to backup controller card 418B as described in reference to Figure 4 above.

[0034] At block 512, method 500 automatically translates the active state associated with the old software image to the new active state associated with the new software image using the metadata. In one embodiment, method 500 compares the metadata associated with the old software to the received metadata to form the differences between the two sets of metadata. With these differences, method 500 recognizes field changes that are compatible for the automatic translation. Method 500 propagates known fields from the current active state to the new active state. For example and by way of illustration, method 500 translates an old active state data structure *A* containing one unsigned int field to a new active state data structure *A* containing a sixteen character string field and the same unsigned int field by offsetting the unsigned int field the appropriate number of bytes to store the new character string field. Method 500 handles fields that are in a data structure of the new metadata by adding, removing, or rearranging fields within the data structure using general purpose, metadata-driven translation routines. In addition, method 500 can assign a default value to a field, increase the size of a field, change an enumeration definition, split a field

into sub-fields, etc. This reduces the amount of specific translation routines, such as the type used in block 514 below.

[0035] At block 514, method 500 optionally manually translates parts of the old active state information that were not translated automatically in block 512. In one embodiment, method 500 uses the old metadata as an input to determine and execute any additional steps needed to translate the old data structure to the new data structure.

[0036] At block 516, method 500 synchronizes the checkpoints of the changing active state during the translation. In one embodiment, method 500 monitors the translation process via a checkpoint mechanism. In this embodiment, each component of method 500 that translates data, reports when the data has been successfully translated. In one embodiment, method 500 waits until each component reports that the data has successfully transferred and translated.

[0037] At block 518, method 500 switches control of the network element from the active control card to the backup control card. In this embodiment, the backup control card is becomes the active card and the former active control card is the backup. In this embodiment, method 500 reboots the former active control card, so that the former active control card is running the new software image. In addition, method 500 synchronizes the state between the new active control card and the new backup control card. For example and by way of illustration, method 500 switches over control from controller card 418A to controller card 418B (as described in reference with Figure 4). Method 500 reboots controller card 418A so that controller card 418A is running the new image. Moreover, method 500 synchronizes the active state in controller card 418B to controller card 418A.

[0038] At block 520, method 500 upgrades the line card. In one embodiment, method 500 uses the new software image to upgrade the packet processor(s) on each line card. Method 500 resets the packet processors so the packet processors are using the new software associated with the packet processors.

[0039] Figure 6 is an exemplary flow diagram of a method 600 for creating the metadata according to one embodiment of the system. In Figure 6, at block 602, method 600 defines the active state data in the DDL. In one embodiment, method 600 defines the active state as discussed in Figure 4 at block 402 above. At block 604, method 600 creates the metadata from the defined state data. In one embodiment, method 600 creates metadata by parsing the state defined in the DDL as described further with reference to Figure 4.

[0040] At block 606, method 600 creates the headers for the defined state data. In one embodiment, method 600 creates the headers as described in Figure 4. At block 608, method 600 compiles the new software from the headers and the software code. In one embodiment, method 600 compiles the new software from the headers and software code as described in reference with Figure 4.

[0041] Figure 7 is a block diagram illustrating a controller card 702 according to one embodiment of the invention. In Figure 7, comptroller card 702 comprises control module 704, upgrade module 706, state receiving module 708, state translating module 710, synchronize state module 712, synchronize checkpoint module 714, new software module 716, and line card upgrade module 718. While in one embodiment, controller card 702 is controller cards 418A-B, in alternate embodiments, controller card is one of controller card 418A-B. Control module 704 performs control of controller card 702. Upgrade module 706 controls the network element software upgrade process of controller card 702. State metadata receiving module 708 receives the new metadata as described in Figure 5, block 510. State translating module 710 translates the received active state to support the new software as described in Figure 5, block 512. In one embodiment, state translating module 710 is translation module 420. Synchronize state module 712 synchronizes the active information between the active controller car and the backup controller card as described in Figure 5, block 510. Synchronize checkpoint module 714 synchronizes the checkpoints during the active state translation as described in Figure 5, block 516. New software module 716

receives and stores the new software for a controller card as described in Figure 5, blocks 504 and 506. Line card upgrade module 718 upgrades the line card as described in the Figure 5, block 520.

[0042] This implementation of the metadata driven upgrade is an example, and not by way of limitation. Thus, network elements having other architectural configurations can incorporate embodiments of the invention. Examples of other network elements that could incorporate embodiments of the invention could have multiple controller cards or have a single controller card incorporating the functionality of metadata driven in-service upgrade. Moreover, a network element having the heuristic and/or DPI peer detection functionality distributed across the traffic cards could incorporate embodiments of the invention.

[0043] Controller cards 204A-B as well as line cards 202A-N included in the different network elements include memories, processors and/or Application Specific Integrated Circuits (ASICs). Such memory includes a machine-readable medium on which is stored a set of instructions (i.e., software) embodying any one, or all, of the methodologies described herein. Software can reside, completely or at least partially, within this memory and/or within the processor and/or ASICs. For the purposes of this specification, the term "machine-readable medium" shall be taken to include any mechanism that provides (i.e., stores and/or transmits) information in a form readable by a machine (e.g., a computer). For example, a machine-readable medium includes read only memory (ROM); random access memory (RAM); magnetic disk storage media; optical storage media; flash memory devices; electrical, optical, acoustical or other form of propagated signals (e.g., carrier waves, infrared signals, digital signals, etc.); etc.

Alternative Embodiments

[0044] For example, while the flow diagrams in the figures show a particular order of operations performed by certain embodiments of the invention, it should be understood that such order is exemplary (e.g., alternative embodiments may

perform the operations in a different order, combine certain operations, overlap certain operations, etc.)

[0045] While the invention has been described in terms of several embodiments, those skilled in the art will recognize that the invention is not limited to the embodiments described, can be practiced with modification and alteration within the spirit and scope of the appended claims. The description is thus to be regarded as illustrative instead of limiting.

CLAIMS

What is claimed is:

1. A computerized method, comprising:

receiving new software to be installed on a network element having an active and backup controller card, wherein the new software comprises a new set of metadata;

installing the new software on the backup control card;

synchronizing an active state of the network element on the backup controller card, wherein the active state is associated with current software running on the network element; and

translating the active state to a new active state associated the new software using the new set of metadata;

2. The computerized method of claim 1, wherein the new set of metadata represents the structure and fields of the new active state associated with the new software.

3. The computerized method of claim 1, wherein the new software is one of a network element operating system and a service running on the network element.

4. The computerized method of claim 1, wherein the translating further uses an old set of metadata associated with an old set of software running on the active control card.

5. The computerized method of claim 1, further comprising:

switching over control from the active controller card to the backup controller, wherein the backup controller card is running the new software; and

running the new software in the active controller card.

6. The computerized method of claim 1, wherein the new set of metadata represents the structure and fields of the new active state.

7. The computerized method of claim 1, wherein the translating supports one or more of adding a field to an active state data structure, removing the field, modifying the field, rearranging fields, assigning a default value, and change an enumeration definition.