APPARATUS AND METHODS FOR PROVIDING SELF-CONFIGURING COMPUTER NETWORKS

A self-configuring network according to a preferred embodiment of the invention comprises a software component that allows devices on the network to take control over their configurations. This software component allows devices on the LAN to become functional members of the network without manual configuration of their network components. It does this by providing the network component of each device with means to obtain values for its own configuration parameters (also known as "configuration objects") from other devices on the network.
— of inventorship (Rule 4.17(iv)) for US only

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.
APPARATUS AND METHODS FOR PROVIDING SELF-CONFIGURING
COMPUTER NETWORKS

Cross Reference to Related Application

[0001] The benefit of the filing date of United States application No. 60/330,536 filed on 24 October 2001 is claimed herein.

Technical Field

[0002] The invention relates to self-configuring computer networks, and more particularly to apparatus and methods for enabling devices to automatically configure themselves when connected to a self-configuring network.

Background

[0003] The rapid growth of Internet Protocol (IP) based computer networking in recent years has led to a wide variety of network arrangements. Network configuration is often problematic. Most, if not all, machines on the network have to be configured manually by a network administrator this is time consuming. Each machine may have different configuration requirements and procedures, making the process error-prone.

[0004] Figure 1 is an example of a typical prior art Internet setup. The global network can be divided into two realms: the Internet and the local area network (LAN) of interest. A gateway connects the two realms together and mediates their communication. Building the network initially requires that all of the devices connected to it be configured properly. Furthermore, as devices are added to and removed from the network, the rest of the devices must be re-configured accordingly so that the network as a whole continues to function. These duties of configuring and maintaining a network are typically left to a trained network administrator. Even when configured by a trained network administrator, networks may contain configuration errors that result in reduced network performance. The network administrator must configure all machines on the LAN, the gateway, and any machines in the Internet realm that are under his control. As devices are added to and removed from the network, the administrator must reconfigure the network to work properly.
[0005] With the anticipated growth of Internet appliances and home networking, the complexity, size, and the variety of devices on modern networks are expected to increase dramatically. This growth will make manual network configuration even more challenging and, in cases like home networking where using a network administrator is not an option, may represent an insurmountable barrier to the proliferation of the market.

[0006] As shown in Figure 2, a generalized network typically consists of inter-connected devices (e.g. hosts, gateways, and network appliances). Devices have requirements that are met by services (e.g. DHCP, DNS, and HTTP) available from other devices. These services, in turn, have configuration requirements that are met by the network administrator. The network administrator configures these services by setting configuration parameters (e.g. namespace, port mappings, subnets, the gateway's IP address).

[0007] There exists a need for self-configuring networks wherein network devices can configure themselves and each other with little to no human intervention.

Summary of Invention

[0008] The invention provides methods and apparatus which facilitate the automatic configuration of computer networks. The invention also provides an architecture for self-configuring networks. The architecture is flexible enough to allow many different implementations.

[0009] A self-configuring network according to a preferred embodiment of the invention comprises a software component that allows devices on the network to take control over their configurations. This software component allows devices on the LAN to become functional members of the network without manual configuration of their network components. It does this by providing the network component of each device with means to obtain values for its own configuration parameters (also known as "configuration objects") from other devices on the network, and help other components to obtain values for their configuration parameters.
[0010] Self-configuring network architecture according to the invention preferably automates configuration tasks, enables communication between services within a realm and between realms and provides a framework to handle exceptions including the insertion, removal, or failure of a device. In addition, it is desirable that the architecture provide network security to ensure the integrity of the network configuration and the network as a whole, and flexibility in the arrangement (topology) of the network and the variety of devices that it will support. The architecture can preferably also accommodate non-self-configuring components.

[0011] Self-configuring networks according to the invention permit network management to be standardized. This has several desirable implications. Such networks will be substantially free of configuration errors, robust, modular, and user-friendly.

[0012] The apparatus and methods provided by the invention can function to make network components configurable with reduced or no human intervention. Devices are able to communicate with each other using any type of transmission protocol and exchange configuration parameters in order to automatically configure themselves on a network. The system adapts when new devices are added, and remains stable when devices are removed or if devices fail.

[0013] In addition, a self-configuring network according to the invention provides for secure network access over any possible topological arrangement. This architecture is flexible and extensible so as to allow the self-configuring network to be distributed. Non-self-configuring network devices may also be accommodated.

[0014] The invention also provides a general framework that can be used for purposes other than a self-configuring network. This architecture allows network entities to exchange information with each other in terms of functionality objects. For instance, the self-configuring network core described herein can be used as a core component of any peer-to-peer solution.

Brief Description of Drawings

[0015] In drawings which illustrate non-limiting embodiments of the invention: Figure 1 illustrates a typical prior art manually-configured network;
Figure 2 illustrates a generalized prior art manually-configured network;
Figure 3 illustrates a portion of a self-configuring network architecture according to the invention;
Figure 4 schematically depicts the transformation of a prior art, manually configured device to a device enabled to function on a self-configuring network according to the invention;
Figures 5 and 6 illustrate the structure of a SCN peer software component according to a preferred embodiment of the invention;
Figure 7 shows two state machines negotiating from their initial states to a final state;
Figure 8 shows an IP packet with four layers;
Figure 9 graphically depicts a set of routing mechanisms that cover the network application classification space;
Figure 10A shows three basic layers of a self-configuring network system according to the invention;
Figure 10B shows a further breakdown of the kernel layer of Figure 10A;
Figure 11 shows four distinct interactions between a self-configuring network device and a self-configuring network system according to the invention;
Figures 12A and 12B show a one-way trusting relationship and a bi-directional trusting relationship, respectively;
Figure 13 schematically depicts an ad hoc network;
Figure 14 shows an ad hoc network similar to that of Figure 13, except that there is a central security entity that handles authentication and authorization;
Figure 15 shows a network with a paranoid security entity;
Figure 16 shows a network wherein the camera is paranoid to the security entity;
Figure 17 shows a network comprising only a topology manager;
Figure 18 shows the network of Figure 17 with a gateway added thereto;
Figure 19 shows the network of Figure 18 with a subnet added thereto;
Figure 20 shows the network of Figure 19 with another gateway and subnet added thereto; and,
Figure 21 shows the network of Figure 20 with still another gateway and subnet added thereto.
Description

[0016] Throughout the following description, specific details are set forth in order to provide a more thorough understanding of the invention. However, the invention may be practiced without these particulars. In other instances, well known elements have not been shown or described in detail to avoid unnecessarily obscuring the invention. Accordingly, the specification and drawings are to be regarded in an illustrative, rather than a restrictive, sense.

[0017] Figure 3 illustrates a portion of a self-configuring network architecture according to the invention, comprising three major components at its highest level: services, self-configuring network (SCN) peers, and transmission protocol. Services are software components running on devices. The SCN peers are software components that coordinate the self-configuration of services. The self-configuring network architecture provides capabilities that would otherwise be assigned to the network administrator. Transmission Protocol provides a common language for communication between SCN peers.

Services

[0018] Services are software components, running on devices, that provide functionality or data to other network services. Individual services may enable communication, provide user-friendly naming conventions, protect data from unauthorized access, etc. From an object-oriented point of view, a service is a software component that exports methods, events and properties. One can execute methods, to access parameters, and to subscribe to events that the service provides. This collection of methods, events and properties is called the "functionality" of the service. In order for a service to function properly, it has to be configured, which entails setting or altering the configuration parameters of the service.

[0019] In a self-configuring network according to a preferred embodiment of the invention, services can configure themselves and obtain configuration parameters from other services. To accomplish this, the invention provides the services with a new functionality, referred to herein as "configuration functionality" (or "configuration objects"). The configuration functionality, which may be included in peer software included in each device, enables a service to configure itself. The
configuration functionality of a service may also help other services with their configurations. The invention provides a general mechanism for enabling services to function in a self-configuring network and identifies preferred network services that are desirable for user-friendly and secure networks. Configuration objects are preferably leased in order to prevent resource depletion, i.e. they are only allocated for limited periods of time.

[0020] Figure 4 schematically depicts the transformation of a prior art, manually configured device to a device enabled to function on a self-configuring network according to the invention. The task of making a network service self-configuring may be viewed notionally as:

1. Adding a SCN peer software module, which handles the service's negotiations with the rest of the network;
2. Replacing the network administrator with a set of configuration functionalities that configure a service via its application programmer interface (API);
3. Moving the configuration parameters from a central location under the direct control of the network administrator and distributes them to the SCN peers of the appropriate services to be owned by the respective SCN peers; and,
4. Recognizing essential network services and enhancing them with auto-configuration functionality.

[0021] A service whose configuration requirements are met by another service is said to import configuration functionality. A service that provides configuration functionality to another service is said to export configuration functionality. An "exporter" is a service that exports configuration functionality. An "importer" is a service that imports configuration functionality. Services may be both importers and exporters of configuration functionalities.

[0022] Every service should be able to communicate its configuration requirements and functionality to all the other services in a standardized way known as a "schema". This may be done, for example, through the use of XML schemas. Exporters communicate their configuration functionality using functionality schemas. Importers communicate their requirements using requirements schemas.
[0023] A functionality schema is a formal definition of an exporter's interface to other services. Other services which know this interface can interact with the exporter to make requests for its configuration functionality.

[0024] A requirements schema is a formal description of the configuration requirements of a particular service. A self-configuring network according to the invention compares the requirements schema of an importer with the functionality schemas of exporters to determine which, if any, of the exporting services can fulfill the requirements of the importer.

SCN Peers

[0025] Figures 5 and 6 illustrate the structure of a SCN peer software component according to a preferred embodiment of the invention. The SCN peer contains the basic self-configuring network operations and information. It is a peer in two senses: all SCN peers are on the same communication abstraction layer; and, all SCN peers are treated in the same way by all other SCN peers.

[0026] The self-configuring network core is a main component of the self-configuring network architecture. Its primary task is to manage the import and export of configuration objects for a service. All SCN peers can have the same core. The details of a preferred core architecture are described below. Interfaces provide means for the core to interact with everything else. Each SCN peer has a protocol interface which enables the core to interact with the network, and a service interface which enables the core to interact with services on the device running the SCN peer.

[0027] The protocol interface removes the details of communicating with the network protocols from the self-configuring network core. It multiplexes and de-multiplexes messages between the self-configuring network core and the network protocols based on a protocol ID. The self-configuring network core is unaware of which protocol(s) will be used on the network; it passes the same messages to the protocol interface no matter what protocols are available. It is the responsibility of the protocol interface to inspect the protocol ID in the message and route the message to the correct transmission protocol. The protocol interface also accepts messages from the protocols and sends them to the self-configuring network core.
The protocol interface can simultaneously support any practical number of protocols; the only limitations are imposed by hardware (e.g. storage space).

[0028] Every self-configuring network needs at least one transmission protocol in order to locate remote services (service discovery) and to pass messages across the network. The protocol interface accepts standardized outgoing messages from the self-configuring network core and translates them into the language of one of the protocols. It also performs the reverse process of translating incoming events from the protocol’s language to the standardized messages of the self-configuring network core. Since the translations are protocol-specific, the protocol interface is also preferably protocol-specific.

[0029] While it is quite possible to use any mechanism as long as the protocol interfaces exchange proper messages, there exist protocols that already are designed for such applications. For example, Universal Plug and Play, Jini™ and Bluetooth™ can be used to exchange messages.

[0030] Services within the self-configuring network framework should preferably be able to locate services that meet their individual requirements. The required services may be located anywhere on the network. A protocol that can speak with all network devices to determine the available services or one that allows a central service repository is suitable for self-configuring network service discovery. Services also need to be able to advertise their existence and their interface to other services on the network.

[0031] Services should preferably have a mechanism to send requests and responses to each other. A remote procedure call (RPC) system is desirable where a function with several parameters can be invoked remotely and where several return parameters can be passed back.

[0032] A service may require the ability to unicast, multicast, or broadcast asynchronous events. In addition to describing the nature of the event itself, the contents of the event may contain other information.

[0033] The service interface removes the details of communicating with local services on the device from the self-configuring network core. It multiplexes and
de-multiplexes messages between self-configuring network core and the services based on a service ID. The self-configuring network core does not know the specifics of the service that the message is destined for. The semantics of the message are interpreted by the service itself. The service interface also accepts messages from the service and sends them to the self-configuring network core. The service interface can simultaneously support any practical number of services; the only limitations are imposed by hardware (e.g. storage space).

[0034] The service interface translates between each service and the standardized messages of the self-configuring network core. As with the protocol interface, the translations that the service interface performs are service specific, so the service interface is preferably also service specific.

The Self-Configuring Network Core

[0035] The self-configuring network core is a main component of the architecture. It provides auto-configuration functionality for all self-configuring network enabled services on its device. The self-configuring network core has two main tasks. The first task is to configure services on its own device. The self-configuring network core does this by importing configuration functionalities (also known as “configuration objects”). The second task is to help configure services on other devices. The self-configuring network core does this by exporting configuration functionalities to cores on other devices. The self-configuring network cores use transmission protocols to communicate with each other.

[0036] The self-configuring network core may be implemented as a finite state machine. It reacts to incoming messages from the protocols and the services modules by changing its state and sending out messages to the protocol and service modules. These messages affect the self-configuring network cores running on other devices in the realm.

[0037] Self-configuring network cores manage imported and exported configuration objects. The imported configuration objects are leased to local services and the exported configuration objects are leased to both local and remote services.
[0038] Network configuration can be viewed as a distributive computation, which is based on external and internal data. The formal framework for such computation is described in Canadian patent application No. 2,357,444 filed on 13 September, 2001, which is incorporated herein by reference. Following this framework we decompose the computation into an external (communication) part and internal (computational) part. The communication part may be factored into communication channels. Each service may implement a finite state automaton (self-configuring network peer) realized as a Mealy machine (a finite automaton in which each state transition is associated with an output) such that the outputs of one machine form the inputs of another, and vice-versa.

[0039] The self-configuring network core corresponds to the external (communication) part and the service interface corresponds to the internal (computational) part. The communication part uses a generic n-stage negotiation scheme to negotiate configuration objects, which can be also viewed as resources, between services. Moreover, the communication part is factored into communication channels, one for each configuration object.

**Object Identification**

[0040] To uniquely identify configuration objects as well as their exporters and importers within a realm, a self-configuring network according to the invention may recognize a set of address information. The address information may include, for example:

- device ID - uniquely identifies a device within the realm;
- service ID - uniquely identifies a service within a device;
- object ID - uniquely identifies a configuration object within a service;
- object Name - is a well-known name for a configuration object imposed by one of the service discovery protocols (SDP); and,
- protocol ID uniquely identifies a protocol mechanism (e.g. Universal Plug and Play (UPnP) or Jini).

[0041] The address information provides an ID that uniquely identifies a configuration object within the realm. The ID specifies the address of every message. The semantics of every message may be in the form of a schema, as described above.
Negotiation

[0042] In a generic n-stage negotiation scheme using a state machine, each state of
the state machine may be given an integer identifier and the negotiation between two
machines takes each machine through its states in order as determined by their
identifiers. Figure 7 shows two state machines negotiating from their initial states
to a final state.

[0043] The following points are important to note in this example:

1. The initial state of each configuration object is different, the initial state 1
initiates the negotiation and it corresponds to an import. The export has the
   corresponding initial state 0.
2. The integer identifier of each state increases by one for each message passed
   between the cores.
3. Each round of negotiation is defined by a message and response. In the first
case, there were two phases and six states numbered 0-5. Hence, an n-round
negotiation includes the states \( Q = \{0, 1, \ldots, 2n+1\} \).
4. The incremental negotiation means that the states in the machine increase by
twos from the initial to the final stage. In particular, one core will always
   have even states (the exporter, or resource provider) and one will have odd
   states (the importer or resource consumer).

[0044] The negotiation framework is useful in itself, but it clearly needs to be
augmented with processing at each SCN peer that handles the specific tasks
associated with the negotiation. To this end, the state machine at each SCN peer is
actually a sum of two machines: the machine handling the negotiation (which always
behaves the same and corresponds to self-configuring network core) and the
machine handling the specific processing tasks (which corresponds to
self-configuring network service interface). These elements are referred to herein as
external and internal, respectively.

[0045] The transition function is thus a sum of the transition functions of these two
state machines. The external transition function returns an integer value that
describes the next stage for the machine. In general, the internal transition function
should return zero and allow negotiation to proceed. However, if an error occurs
and the internal processor needs to back the negotiation up, it may do so by producing a negative value and causing a regression to an earlier state.

[0046] The internal elements of the state machine are left to be defined by the service provider. The internal transition function should preferably return an error level that moves the state machine back to the appropriate state from which to re-start negotiation. The service provider must keep in mind that the output of this function is added to the output from the external transition. The internal output function should preferably produce an output that is understandable by the negotiating peer and that will force that peer to transition to an earlier (or lower) state.

Leasing and Robustness

[0047] Robustness can be introduced to this framework by adding an $\epsilon$-transition that is executed periodically. The $\epsilon$-transition moves the negotiation back a stage every time it is executed. This property introduces fault tolerance. Each time a timer expires and executes the $\epsilon$-transition, the negotiation steps toward its initial state. If both negotiating parties are alive, this transition will simply force a re-negotiation of the last stage. If one of the cores is not accessible from the other (e.g., a component or link failure), the remaining machines will begin a march to their initial states. If a core state machine reaches its initial state, the "lease" can be considered to have expired.

[0048] This mechanism also provides generic fault recovery. Because of the incremental nature of the negotiation, two negotiating parties will always fall back to the earliest appropriate stage of negotiation. For example, assume that the $\epsilon$-transitions have moved two cores involved in a complex negotiation back several steps due to a link failure. When the link is re-established, the two parties will send each other information about their current state. The core with the higher state will immediately move to the appropriate lower state in order to re-start negotiation.

[0049] For each $\epsilon$-transition the state machine moves back two states (by state identifier value) and announces this new state to its negotiating partner. The fact that the machine moves back by two states is a result of the incremental nature of the negotiation.
Deadlock and Starvation

[0050] We will now discuss a more complex system in which multiple automata are negotiating among one another in an effort to reach a final state. Two problems can arise in such a system:
1. Deadlock can occur if there is a cyclic dependency between the automata.
2. Starvation can occur if there are not enough resources for all of the automata to reach a final state.

[0051] Each automaton in the system can be factored into one or more constituent automata that correspond to each negotiation channel. These negotiation channels have two ends: one requesting a resource and one that provides the resource. As a result, each of these constituent automata behaves like a resource provider or resource consumer as described above. For example, automaton $A'$ may be associated with three negotiations, and would thus be the product of these automata: $A' = A'^1 \times A'^2 \times A'^3$. Understanding this breakdown and the numbering scheme used to describe it will be important for the following discussion. The following paragraphs will outline how starvation and deadlock can be detected and, if possible, prevented.

[0052] By analyzing the states of all of the automata involved in the system-wide negotiation, we can compute several interesting values, the first of which is given here:

$$u = \Sigma_i (-1)^i \lfloor q_i/2 \rfloor,$$

where $i$ counts each automaton, $j$ counts each negotiation channel associated with the given automaton, and $\lfloor x \rfloor$ denotes the integer part of $x$. $q_j$ thus indicates the state of the $j$th constituent automaton of the automaton $A'$, i.e., the constituent automaton $A'^j$.

[0053] It can be shown quite easily that after each round of negotiation, $u = 0$. This gives us a first important criterion for the overall state of the system: if $u$ deviates from 0, the negotiation protocol is not implemented properly.

[0054] The second interesting value can be computed by taking the following sum:

$$r = \Sigma_i \lfloor q_i/2 \rfloor$$
[0055] If all of the automata implement \( n \) rounds of negotiation and there are \( m \) total negotiations (or channels), then the above sum yields \( r = 2mn \) when all of the automata have reached their final state. This gives us a second criterion for the system state: \( \text{if } r < 2mn \text{ for a long period of time, a deadlock exists within the system.} \) The intuition here is that if \( r < 2mn \) then there is at least one negotiation that has not reached its final state. If this condition persists, it is safe to assume that the negotiation is being prevented from completing due to a deadlock in the system.

[0056] Detecting starvation is fairly straightforward, but it is facilitated by the following behavior from the resource providers and consumers in the system:

1. Each provider caches all potential consumers of its resource, and vice-versa.
2. Providers and consumers each randomly chose a partner for negotiation from this cache.
3. If a resource consumer fails to get a resource from a provider, it will randomly chose another provider from which to attempt to get the resource.

[0057] With these augmentations, starvation is detected as follows: if a consumer has an empty cache of the possible providers, the consumer will be prevented from having the resource and will thus be starved.

[0058] The following is an example of the steps of the self-configuring network core task:

1. On a service startup, the service installs state machines for its imports and exports. The imports are initialized to 1 and exports are initialized to 0. The imports are required configuration objects, which the service needs to function properly and the exports are objects, which can help in configuring of other services.
2. The initialization triggers the discovery phase of the configuration objects.
3. When the configuration objects are discovered, the external machines pass the discovery phase and information about location of the required configuration objects is added to the state data.
4. Consequently self-configuring network core can use next rounds of negotiation to authenticate and authorize the importer, and finally to acquire the required configuration objects following an incremental negotiation.
5. When the configuration object is acquired, lease information is associated with it. The state data will include the added lease information and the
location of the providing service. The lease will naturally time-out and the acquisition will have to be renewed.

[0059] When all requirements are satisfied, the service runs properly, provides the full set of its exported configuration objects, and fulfills requests from other services.

Exception Handling

[0060] The services, the core, or the transmission protocols may generate exceptions, which will be logged on the devices that generated them. Furthermore, depending on the context of the exception, all other self-configuring network services will be notified of the exception. In most cases, it is the responsibility of the service to handle the exception properly.

[0061] Some exceptions will occur undetected. For example, if an exporter is removed ungracefully, the importing service will simply malfunction. The importer will naturally time out and try to discover another exporter. If an exporter cannot be discovered, then the importer must keep trying and keep working as best as it can.

[0062] If a service restarts, for example from a previous crash or reboot, it will enter the initial phase. The other services will discard their leases in response to this and try to renegotiate.

[0063] When the network connection is lost, things will malfunction and it will appear as if some services are not available. There is no choice but to wait for the network to come up. If the protocol interface is unable to deliver a message, it will maintain it in a queue for periodic retry until the message can get through, with a provision for eventually giving up (and making a note of this in the log).

Desirable Properties for a Self-Configuring Network

[0064] This section outlines some preferable functional properties for automating network configuration and maintenance. A self-configuring network according to the invention may have some or all of these properties.
[0065] A fundamental requirement of self-configuring network is that these services can be discovered and used by network applications without any a priori knowledge of the network configuration. Each service offered in a self-configuring network should preferably be discoverable by network applications without a priori knowledge of the network configuration.

Connectivity

[0066] A self-configuring network is preferably able to manage required resources to allow devices on the LAN to intercommunicate and to communicate with devices on a wide area network (WAN) such as the Internet. Both the LAN and WAN are considered to be connected IP address spaces, which means that managing IP routing resources is both sufficient and necessary to guarantee connectivity in both realms. This yields the first two properties:

Property 1: A self-configuring network should preferably manage IP addresses, subnets, and gateways on the LAN.

Property 2: A self-configuring network should preferably acquire and manage a pool of one or more addresses in the WAN address space. A self-configuring network should preferably maintain routing information for each address.

Traffic Management

[0067] The Internet Protocol assumes that all devices with IP addresses are accessible using only basic IP routing. In reality, technologies like network address translation (NAT), which allows multiple devices within a LAN to share a limited number of IP addresses, and firewall packet filtering have broken this assumption. The result is an Internet filled with disparate address realms. Communication between realms requires the explicit intervention of a higher-layer protocol or a trained system administrator at the gateway separating the realms.

[0068] A self-configuring network may offer a traffic management facility for devices to configure the gateway routing mechanisms to meet their requirements. The self-configuring network classifies applications by specifying integrity and
access constraints. An integrity constraint tells the gateway how much of a packet to leave untouched while routing. An access constraint tells the gateway how the application wishes to be accessed. The access and integrity constraints can each be described by specifying a layer in the IP packet for each constraint.

[0069] Figure 8 shows an IP packet with four layers. An integrity constraint is described by naming one of the layers. For the constraint to hold, the layer named and all of the layers below it should preferably not change from the source to the destination. For example, if an application has an integrity constraint of "transport", the transport and application layers must remain intact during the packet's travels.

[0070] This classification is arguably too strict: for example, one can imagine a scheme in which two non-joined layers must remain intact, but those in between can change (e.g., IP and application must stay the same, but transport can change). This, however, does not conform to the standard method of stack-based networking in which each layer of the packet is removed before presenting it to the network stack layer above. A network application will typically sit at one layer of the network stack and have integrity expectations based only on the layers that it reads and manipulates.

[0071] Access constraints are described by binding access keys to each of the layers described above. A bound access key is called a routing element because it describes how to get that layer of the packet to a specific destination. An access key is either a fixed value that conforms to the addressing of a given layer (e.g., IP: 24.13.147.88, transport: TCP-80), or a "don't care". Once bound, the routing elements form a routing key that is used by a gateway to route a packet to its proper destination.

[0072] The process of binding access keys to packet layers converts "don't cares" to fixed values. This means that a single layer can be used to identify nature of a routing key; the layer described and all layers above are bound to routing elements.

[0073] Routing keys and integrity constraints are related in the following way: overlapping layers comprise passive routing elements. These elements are necessary to properly deliver the packet, but may not be altered during the routing
of the packet. Layers that do not overlap (because there is no integrity constraint) can be active routing elements, as those packet layers may be altered during packet delivery. For example, a server application may have an integrity constraint of IP, yet specify that a certain transport routing element be used. The packet must be delivered untouched from the IP layer, but the extra access information lets the gateway multiplex numerous servers on a single IP address as long as the transport routing elements don’t collide.

[0074] On the other hand, an application may specify transport integrity, but require that the routing element for the IP layer be a specific value. This could happen with an application that is involved in an active "session" with a remote node (e.g. H.323). The remote node must continue to send packets to a single IP address for the duration of the session, but a gateway may alter the IP layer at any point to deliver it to the proper destination.

[0075] Armed with these definitions, the space of network application requirements can be summarized using the graph in Figure 9. The "application +" element is for applications that allow some degree of application-layer manipulation during transport.

[0076] A self-configuring network preferably offers a set of routing mechanisms that cover the network application classification space, as shown in Figure 9. While the invention allows for many possible routing mechanisms, in a given installation, only a sparse set of routing mechanisms may be required. Some routing mechanisms may rarely, if ever, be required, for example: media access control (MAC) to Application. This leads to the next property:

Property 3: A self-configuring network should preferably offer a suite of routing mechanisms that covers the entire network application classification space.

Naming

[0077] Accessing network resources requires that one know the access mechanisms associated with the resource. A problem with network access mechanisms is that they are typically cryptic to humans and may change frequently in a dynamically
configured network. As a result, a self-configuring network should preferably offer the ability for devices to give resources and services human-readable identifiers. The process of applying an identifier to an access mechanism is called naming and the reverse process is called resolving a name.

[0078] Naming and name resolution requirements exist for both the LAN and WAN—or more generically, exist in all realms in which resource access is desired. The naming properties outlined below highlight the concerns regarding resource naming and name resolution. It is important to note at this point that the domain name system (DNS) is mentioned explicitly in the requirements because it is by far the prevalent name resolution mechanism used by the Internet and on IP networks.

Property 4: A self-configuring network should preferably provide a mechanism to dynamically assign names to device IP addresses within the LAN.

Property 5: A self-configuring network should preferably have a DNS server capable of resolving LAN resource names into IP addresses.

Property 6: A self-configuring network should preferably have a DNS server capable of resolving WAN URLs into WAN IP addresses.

Property 7: A self-configuring network should preferably provide a mechanism to dynamically assign names to routing keys for devices that wish to export their resources into the WAN.

Property 8: A self-configuring network should preferably provide a resolution mechanism for converting names to routing keys that acknowledges the limitations of DNS and takes advantage protocol-specific assumptions where appropriate.

[0079] It should be noted that DNS has the following limitations:

1. DNS resolves names into IP addresses only. This is reasonable since DNS is meant to be an enhancement to IP, but it does restrict the ability to convert a
name into the access mechanism needed to access resources. Hence, an assumption must be made that both ends of an application have specific, implicit expectations with regard to how to access a resource. For example, it is reasonable to assume that an HTTP application will attempt to access an HTTP resource using TCP port 80, even though that information is not explicitly communicated to the client application through DNS.

2. A DNS lookup is not a strict requirement for most network applications and protocols. As a result, it is arguably unreasonable to assume that all applications using network resources will perform a DNS query before trying to access the resource. An understanding of common network communication, however, supports the counter argument. Hence, it will be assumed that all network resource accesses are preceded by a DNS query, with the hope that contrary behavior will be sufficiently minimal as to cause little or no loss in functionality.

Security

[0080] The term security is used to cover a broad spectrum of concerns regarding IP networking. For the purposes of a self-configuring network according to the invention, security will be defined to mean that network resources can be protected from malicious use. Network resources may be those providing self-configuring network functionality, or may be outside of the scope of the self-configuring network such as a video feed from a camera. Self-configuring network security will allow both types of resources to be protected. This leads to the next property:

Property 9: A self-configuring network should preferably provide a mechanism to protect network resources from malicious use through an effective authentication and authorization scheme.

[0081] A natural corollary to this property is that devices that wish to make use of self-configuring network configuration functionality must be able to identify themselves in a way that can be subject to authentication and authorization (A & A):

Property 10: Self-configuring network devices should preferably provide identification that can be subject to authentication and authorization (A & A) when they communicate among each other.
Property 11: The routing mechanisms offered by self-configuring network as a result of Property 3 should not offer more ingress traffic flow than is required by the LAN.

[0082] Security is discussed further below with reference to Figures 12 to 16.

Network Monitoring

[0083] Although one aim of the invention is to automate the process of configuring and maintaining networks, in some cases networks will have unique configurations that require that the policy assumptions made by a self-configuring network according to the invention be adjusted by the network user. As such, a self-configuring network preferably provides a network monitoring service that can be used to query the state of the network and to update network parameters as appropriate.

Property 12: A self-configuring network preferably has a network monitoring facility that allows a user to fine-tune the network configuration to address concerns that are unique his or her network.

System Elements

[0084] The following sections describe various system elements that can be used to address the properties described above. Each element may be associated with one or more protocols that dictate how the services offered by the elements can be found and how the resources offered by the service are to be accessed. A discussion of the protocols that will be used is outside of the scope of this document. For each resource required by an application, the application must interact with the system element as dictated by the protocol. In general, this document does not discuss the application-side element that interacts with the self-configuring network system elements described here, as its existence and behavior is made obvious by the application's requirements and the access protocols used.

[0085] The following table shows how the properties discussed above may be provided by building various system elements into a self-configuring network:
<table>
<thead>
<tr>
<th>Category</th>
<th>Element</th>
<th>Description</th>
<th>Property(ies)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connectivity</td>
<td>Topology Manager</td>
<td>Manages the network topology, stores information that describes topology elements.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>Gateway</td>
<td>Divides IP subnets on self-configuring network LAN.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>LIPM (Local IP Manager)</td>
<td>Manages IP addresses and routing information for devices on a LAN.</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>WAN Address Manager</td>
<td>Manages IP addresses and routing information on the WAN.</td>
<td>2</td>
</tr>
<tr>
<td>Traffic Management</td>
<td>Traffic Manager</td>
<td>Handles traffic management for communication between realms; offers applications a method for making resources accessible in any realm.</td>
<td>3, 8, 11</td>
</tr>
<tr>
<td>Naming</td>
<td>DNS Server</td>
<td>Resolves LAN and WAN names.</td>
<td>5, 6</td>
</tr>
<tr>
<td></td>
<td>LAN Name Manager</td>
<td>Allows applications to &quot;name&quot; resources on a network by assigning IP addresses for LAN to LAN naming.</td>
<td>4</td>
</tr>
<tr>
<td></td>
<td>WAN Name Manager</td>
<td>Allows applications to &quot;name&quot; resources on a network by assigning routing keys for WAN to LAN naming.</td>
<td>7, 8</td>
</tr>
<tr>
<td>Security</td>
<td>LSAM (Local Secure Access Manager)</td>
<td>Handles authentication and authorization for resource access on a device.</td>
<td>9, 10</td>
</tr>
<tr>
<td>CSAM</td>
<td>Handles network-wide authentication and authorization for resource access.</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--------------</td>
<td>-------------------------------------------------------------------------</td>
<td></td>
<td></td>
</tr>
<tr>
<td>(Central Secure Access Manager)</td>
<td></td>
<td>9, 10</td>
<td></td>
</tr>
<tr>
<td>Network Monitoring</td>
<td>Monitors the state of the network and gives the user the ability to adjust self-configuring network policies for the behavior of each system element.</td>
<td>12</td>
<td></td>
</tr>
</tbody>
</table>

**System Elements**

5

[0086] Managing the arbitrary growth of an IP LAN can be accomplished using three distinct system elements: a Topology Manager, one or more Gateways, and a Local IP Address Manager for each gateway. The Topology Manager oversees the growth of the network and ensures that all of the devices in the network are fully interconnected. Gateways link subnets, and Local IP Address Managers manage the pool of addresses allocated to each subnet.

10

**Topology Manager**

15 [0087] The Topology Manager preferably controls all of the IP addresses and subnets available to a LAN. Network elements can allocate IP subnets from the Topology Manager, and the Topology Manager ensures that all of the Gateways in the LAN are properly configured to route packets between subnets. The set of subnets and Gateways comprises the topology of the LAN. The Topology Manager stores this information and uses it to keep the network fully interconnected at all times. The Topology Manager can also store arbitrary descriptors for each Gateway and subnet in the LAN. These descriptors can be used to identify features of network segments—for example, the security system may wish to mark one subnet as trusted and another untrusted.

20

25
Gateways

[0088] Gateways are IP routers that link subnets. They may be used to bridge networks with incompatible MAC addressing or to provide access control between subnets. Gateways monitor the topology stored by the Topology Manager and update their routing information accordingly.

Local IP Address Manager

[0089] Each Gateway has a Local IP Address Manager that manages the pool of IP addresses available to the Gateway's subnet. Hosts on the subnet may request addresses from the Local IP Address Manager, and the Address Manager must provide them with an address and the routing information needed by the host to move packets through the LAN.

WAN Address Manager

[0090] The WAN Address Manager is the system element that controls the WAN addresses and routing information for each address. The addresses controlled by the WAN Address Manager can be allocated dynamically by the Traffic Manager (discussed below) to create the routing keys needed to provide traffic management.

Traffic Manager

[0091] A Traffic Manager may be used to perform traffic management. Every realm that wishes to intercommunicate with devices in another realm must have at least one Traffic Manager. The Traffic Manager will typically be directly associated with the Gateway device that logically separates the realms. Traffic Managers register themselves with the Topology Manager and are added to the network topology in a similar way to Gateways.

[0092] Traffic Managers accept requests from devices that describe the device's integrity and access constraints. The Traffic Manager then selects a routing mechanism depending on the capability of the associated Gateway and binds the access key to a routing key. The Traffic Manager optionally informs the device of the values for "don't care" access keys after binding.
[0093] The routing mechanisms offered by each Gateway should cover the entire classification space as outlined in Property 3. In doing so, a Gateway ensures that any type of application may be run behind it without concern for proper operation. Note, however, that even if the mechanisms provided offer a complete solution set, it is still possible for applications to exhaust Gateway resources and to prevent other applications from working properly.

[0094] Applications generally only associate with a single Traffic Manager—either that of the device’s local Gateway, or the next-closest Traffic Manager between the device’s Gateway and the root Gateway. Traffic Managers are aware of each other and are able to update all of the traffic management rules in a network on behalf of a device. This behavior leaves devices ignorant of the network topology and greatly simplifies the duty of individual devices.

[0095] New Traffic Managers that are added to a network use the traffic management information of their default Traffic Manager to configure themselves with the current traffic management configuration of the network.

[0096] A traffic management session may occur when a device wishes to establish an ongoing communication session with a device across a Gateway with traffic management services. At the time of a traffic management request, a device can optionally specify the address of the remote device for which traffic management is needed. If the default Traffic Manager for the device is not the appropriate Manager for a given session (i.e., because the remote device is accessible across a different Gateway than that associated with the default Traffic Manager), the Traffic Manager can re-direct the device to use the proper Manager. Traffic Management requests are then handled by the new Traffic Manager for the duration of the session.

[0097] One advantage of Traffic Management is the ability to do delayed binding of "don’t care" elements of access keys. This means that access keys are not bound to routing elements immediately, but are only bound at the time that communication using the access key occurs. For example, an HTTP server may request traffic management for TCP port 80, but not specify a fixed IP address when the access key is submitted. The Traffic Manager may choose to delay the binding of the IP
address to a routing key until the HTTP service is actually requested. Upon the completion of the HTTP session, the routing key could be reverted back to the original access key and the Traffic Manager could again wait until the service was requested to bind the access key.

[0098] This process would allow for a much more efficient use of available IP addresses, since IP addresses would only be bound to services when the service was active. Inactive services would remain unbound and would not waste valuable IP addresses.

DNS Server

[0099] All applications on the LAN require access to a DNS server, or equivalent, that can resolve LAN and WAN names to IP addresses.

LAN Name Manager

[0100] Self-configuring network devices preferably use a LAN Name Manager to register one or more names for resolution on the LAN. Because of the access patterns on the LAN, it is sufficient for the LAN Name Manager to assign names to the IP address of the resource.

WAN Name Manager

[0101] The WAN Name Manager provides a mechanism for LAN devices to register names that WAN devices can use to access LAN network resources. The WAN Name Manager interacts closely with the Traffic Manager to bind names to routing keys in an appropriate way, as discussed above.

[0102] The interaction may proceed as follows:

1. A device requests traffic management to export a service onto the WAN. As part of the access key, the device specifies a URI at which the service should be reached: e.g., http://camera.armadillolabs.com.

2. The Traffic Manager recognizes that the URI requires WAN name resolution and registers the URI with the WAN Name Manager.
At this point, the Traffic Manager may have already bound all of the elements of the routing key—including the IP address. If this is the case, it simply gives this information to the WAN Name Manager, and the WAN Name Manager is immediately able to serve name resolution requests from devices on the WAN.

However, the Traffic Manager may be more intelligent and did configure to avoid binding the IP address until necessary. In that case, the following steps would occur:

3. The WAN Name Manager receives a request for name resolution for a LAN resource, but it finds that the IP address has not yet been chosen. It immediately alerts the Traffic Manager.

4. The Traffic Manager binds the IP address to the resource, binds any other access keys that were left unbound (creating the routing key), and returns the IP address to the WAN Name Manager.

5. The WAN Name Manager returns this IP address to the requesting application. The gateway uses the routing key to properly route traffic bound for the given IP address the LAN resource.

Central Secure Access Manager

[0103] The Central Secure Access Manager (CSAM) coordinates authentication and authorization for a self-configuring network. It has a database of entities that are authorized to access subsets of the resources provided in the network and can provide security information for each entity on demand. The entities in the database have capabilities that describe the resources they may access.

[0104] Entities may enter the database in two ways:

1. The CSAM offers a service that allows self-configuring network devices to register their network entity identifiers with it. If it is configured to be trusting, it will believe the registration by default. If it is configured to be paranoid, however, it will not believe the registration and will require a trusted security key to be provided in order for the registration to be accepted. In general, the network operator will provide the trusted key in the form of a passcode. The CSAM may be configured to be both trusting and paranoid depending on the nature of the registering entity. For example, it may be desirable to provide trusted access for devices on a wired LAN segment, and paranoid access to devices on a wireless segment.
An accepted registration does not guarantee that the entity will be granted access to any of the network resources, only that the entity is now known to the network. The CSAM may be configured by default to grant all accepted registrations full access, or the network operator may configure it to offer finer-grained control over resource access.

2. The network operator assigns access privileges to network entities that are not associated with self-configuring network devices (and hence cannot request registration). This may be the case if a user on the Internet wishes to access a protected resource (such as a camera video stream). The network operator could create a network entity for the user and associate a passcode with it so that the user could gain secure access to the resource.

Local Secure Access Manager

[0105] The Local Secure Access Manager (LSAM) handles authorization and authentication on a device. The LSAM is a trusted entity to any other network entities on the device, hence those entities that are configured to be paranoid may use the LSAM to provide authentication and authorization. An LSAM may interact with the CSAM in two ways:

1. It may register the device with the CSAM in order to access resources on the self-configuring network. Clearly this step is not required if the device does not need to access network resources.

2. It may request authentication and authorization information from the CSAM. Note that this step requires that the CSAM trusts the LSAM and thus necessitates that the LSAM register itself with the CSAM in advance.

[0106] If a LSAM wishes to use the services of a CSAM, it may chose to be trusting or paranoid toward the CSAM. If it is trusting, then it will by default believe the security information provided by the CSAM. If it is paranoid, however, it may be made to believe the CSAM with which it is negotiating (through the use of a security key exchange), or it may seek a trusted CSAM. A third, divergent option also exists: a LSAM may decide not to use a CSAM if it cannot find one that it trusts and can elect to handle all authentication and authorization itself. This could be particularly useful if a CSAM is not available, either due to a network failure or because the network was never configured with one in the first place.
Network Monitor

[0107] To address Property 12 above, a self-configuring network system should preferably have a network monitor facility that can monitor the state of the network and that allows users to adjust the network configuration to address the user’s unique requirements. This can be implemented as a single system element—the Network Monitor—that is associated with an intuitive user interface. The exact behavior of the Network Monitor is dependent on the implementation of the other system elements and on the unique requirements of the network in which self-configuring network will be deployed.

System Structure

[0108] The structure of the self-configuring network system can be broken into three basic layers, as shown in Figure 10A. The kernel offers basic connectivity and security to devices. The resource access management layer offers devices the ability to make resources available through the use of naming and traffic management. Network monitoring can best be implemented as an application that will make use of the self-configuring network infrastructure, and as such, is not described in detail herein. This breakdown is useful because it allows the resource access management services to be implemented on separate devices from those that implement the kernel functionality.

[0109] Figure 10B shows a further breakdown of the kernel of Figure 10A. The Local Security layer is preferably the lowest layer in the self-configuring network system, and hence has no dependencies within the self-configuring network system. The lack of dependencies at the Local Security layer eliminates potential race conditions between the security elements and other elements in the system that could weaken the security model. Preferably, the only system element that exists at this layer is the LSAM.

[0110] Connectivity is built on top of the Local Security layer. This allows the connectivity services to be protected before connectivity itself has been established, eliminating an awkward cyclical dependency that would otherwise exist. The Topology Manager, the LIPM, and Gateways may operate at this layer, since they are crucial to establishing connectivity. Once the connectivity layer is established,
separate devices that offer Gateway and LIPM functionality can be added to the network.

[0111] The WAN Address Manager also preferably operates at this layer. It is interesting to note that its functionality is essentially identical to that of the LIPM and as such can be considered to be the same element for the purposes of this model. The differences between the two are based only on the mechanisms that each uses.

[0112] Network Security is established once the LSAM can communicate with a CSAM to share authentication information. To avoid a dependency between the LSAM and the CSAM (which would violate the layer dependency restriction), the Network Security layer has two elements: the LSAM-NET and the CSAM. The LSAM-NET can be considered to be a separate element from the LSAM that allows network-wide connectivity information to be shared. The LSAM-NET augments the LSAM security information, which gives us an appropriate dependency.

[0113] Care must be taken to ensure that the final implementation is free from race conditions and interdependencies that could weaken the security system. In the worst case, every LSAM-NET could chose not to believe the CSAM and leave the LSAM to do its own authentication and authorization which restores the advantage of the stand-alone Local Security layer.

[0114] Once network connectivity is established, devices are preferably able to make their resources available for use, and able to use other devices' resources in the network. These capabilities fall into the broad category of Resource Access Management. The system elements involved in Resource Access Management are those handling the naming requirements (e.g. the DNS) and the Traffic Manager.

[0115] The Resource Access Management layer presents a single interface to devices that allows them to describe how and where to access their resources. The interface is functionally identical to that of the Traffic Manager and as such will be controlled by that element. The Traffic Manager handles the naming requirements of the resource in each realm and establishes the necessary routing keys to properly route traffic to and from the resources.
Dependencies

[0116] The following two tables outline some of the dependencies between elements in the layers described above. Dependencies between layers will be considered to be implicit and will thus not be analyzed in detail.

[0117] This table outlines some of the dependencies between layers of the Kernel shown in Figure 10B:

<table>
<thead>
<tr>
<th>Kernel Layer</th>
<th>Element</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Local Security</td>
<td>LSAM</td>
<td>None</td>
</tr>
<tr>
<td>Connectivity</td>
<td>Topology Manager</td>
<td><em>LSAM</em>: May use LSAM to authorize access to IP connectivity information.</td>
</tr>
<tr>
<td></td>
<td>Gateway</td>
<td><em>Topology Manager</em>: Gateways depend on a Topology Manager to register themselves and to get the topology of the network.</td>
</tr>
</tbody>
</table>
| LIPM             | Gateway       | *Gateway*: If the LIPM is managing a subnet address range, it depends on a Gateway being present to define the subnet boundary.  
                                  |                                                                            | *Topology Manager*: Uses Topology Manager to allocate a range of IP addresses.  
<pre><code>                              |                                                                            | *LSAM*: May use LSAM to authorize access to IP connectivity information. |
</code></pre>
<table>
<thead>
<tr>
<th>Category</th>
<th>Element</th>
<th>Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Naming</td>
<td>Name Manager (LAN &amp; WAN)</td>
<td>Kernel</td>
</tr>
<tr>
<td></td>
<td>Name Resolver (DNS Server)</td>
<td>Kernel</td>
</tr>
<tr>
<td>Traffic Management</td>
<td>Traffic Manager</td>
<td>Kernel: In particular, requires a Gateway to establish routing keys for resource access. Name Manager: Uses name management to establish resource presence in appropriate namespaces. Name Resolver: May use a name resolver for delayed binding.</td>
</tr>
</tbody>
</table>
The Device's Perspective

[0119] A self-configuring network system offers a simple and powerful tool for devices to connect to and interact within an IP network. A device sees only three basic service classes that address the properties discussed above:
1. Security: All devices are capable of protecting their resources through the self-configuring network security mechanism. A local secure access manager (LSAM) allows devices to secure resources even before network connectivity has been established. Once network connectivity is established, a central secure access manager (CSAM) can be used to augment the security information used by the LSAM.
2. Connectivity: Devices will be given the configuration information needed to intercommunicate with other devices on the network from a local IP manager (LIPM). Special devices may extend the network by using the resources available from a Topology Manager.
3. Resource Access: Through a Traffic Manager, devices can explicitly define how their resources are to be accessed and how they wish to access other resources. The Traffic Manager configures the naming and routing systems on the network to meet the device requirements. The naming system allows devices to resolve resource names into the actual address of the resource.

[0120] In most cases, these classes manifest themselves as four distinct interactions between a self-configuring network device and the self-configuring network system elements described in this document, as shown in Figure 11.

Security Model

[0121] Based on the above definition of security, it is possible to develop a model of security that is sufficient for the needs of a self-configuring network. An important distinction will be made throughout this discussion between security mechanisms and security policies. Mechanisms do the actual work, policies are the set of rules that the security system implements.

[0122] Security mechanisms that may be provided to users of a self-configuring network system are described below. At this level of abstraction, the mechanism-policy divide is clear and it is up to the users of self-configuring
network to provide the set of rules that will be followed for the system. The divide between mechanism and policy depends on perspective, however, and it is important to note that the model can also be viewed as a security policy that must be followed by the underlying security mechanisms used by self-configuring network. In other words, the security model described should be viewed as a mechanism if the reader is a user of self-configuring network and should be viewed as a policy if the reader is implementing the security system of self-configuring network.

[0123] The self-configuring network security model describes how network entities identify themselves to each other and the different relationships that are established as a result. The term "network entity" is used to identify any application or device on a network that has an identifier that can be subject to authentication and authorization.

[0124] Property 9 preferably drives the behavior of a self-configuring network security system. The key to the model is the establishment of trusting relationships between network entities. A trusting relationship is one in which a network entity believes information provided by another. The information that is sent can be data or it can be a request for access to a resource. In either case, the trusting entity will react to the information as appropriate (for example, it will grant access to the resource if the information was a resource request).

[0125] Note that the relationship between two entities is conceptually two-way, hence a trusting relationship may exist in either one or both directions. Figures 12A and 12B show a one-way trusting relationship and a bi-directional trusting relationship, respectively. The arrows point from the trusting entity to the entity that it trusts, e.g., in the Figure 12A A trusts B but not vice versa, and in Figure 12B A and B trust each other.

[0126] All relationships between network entities begin with an exchange of entity identifiers. Network entities can be categorized as either trusting or paranoid depending on their treatment of new identification information from another network entity. A trusting entity trusts all other network entities by default, and as a result a trusting entity always believes information provided by other entities. A paranoid entity, on the other hand, does not trust other network entities by default.
and must be provided with authorization and authentication of a new identifier from a
trusted third party before it can establish a trusting relationship.

[0127] The following question arises: how does a paranoid entity ever learn to trust
any other entity? There are two approaches.

1. The paranoid entity may exist on a device with an authentication and
   authorization entity that it innately trusts. It can then defer to the judgement
   of the authentication and authorization entity about whether the new
   identifier belongs to a trusted entity.

2. The paranoid entity may request that the new entity provide it with a trusted
   key to prove that it can be trusted. In general, the key would comprise a
   passcode of some sort that is unique to the paranoid entity. A user on the
   network could enter the passcode on behalf of the external entity in order to
   establish the trusting relationship.

[0128] Entities may be given finer-grained behavior with respect to the concepts of
trust and paranoia. For example, it may be desirable for an entity to be trusting of
other entities with an IP address from a trusted range (e.g., the LAN) and to be
paranoid of entities with other IP addresses (e.g., the WAN). Or an entity may be
trusting of other entities from which it is requesting a resource, but paranoid
towards other entities that are attempting to access its resources.

[0129] To summarize, the self-configuring network security model has the following
features:

1. Network entities preferably have identifiers that can be subject to
   authentication and authorization (A & A).

2. All relationships between entities preferably begin with an exchange of these
   identifiers.

3. If an entity trusts another entity, it believes all information from the other
   entity. This is a trusting relationship. Trusting relationships are directional.

4. Trusting entities trust other entities by default, hence they believe
   information received from other entities by default. Information in this case
   can be anything from data to a request for a resource.

5. Paranoid entities do not trust other entities by default, and must be convinced
   to believe the other entity. This can be done by deferring to another trusted
   entity for authentication and authorization, or by providing the paranoid
entity with a trusted security key. Once a paranoid entity is convinced to believe another entity, the paranoid entity trusts the new entity.

[0130] Figure 13 schematically depicts an ad hoc network. An ad-hoc network is preferably not encumbered by the security system in place. Ideally, there should be no configuration involved in building such a network. Nonetheless, there may be a requirement that some resources are protected by an authentication and authorization (A & A) scheme. In this example, the network consists of a self-configuring network gateway and a self-configuring network camera. The self-configuring network configuration resources are preferably trusting network entities so that the network configuration can proceed without intervention. The camera video stream, however, is paranoid and defers to a trusted authentication and authorization entity on the camera itself. This entity can be configured in any way desired: it may have a fixed set of passcodes that grant access to the video stream, or it may be user-configurable to have a set of users that are allowed to view the video stream.

[0131] Figure 14 shows an ad hoc network similar to that of Figure 13, except that there is a central security entity that handles authentication and authorization. Devices can register themselves with the security entity in order to be granted access to resources on the network. This registration process is no different than any other resource access—the central security entity would have to trust the registering device before it would accept the registration. In this case, the security entity is trusting to devices on the registering device’s network segment and it thus accepts the registration. What the security entity does with the registration is not determined by whether it is trusting or paranoid and is left as an implementation detail. (For example, it may classify registering devices and determine which services they should be allowed to access.)

[0132] In this network, the camera’s authentication and authorization entity is trusting and hence trusts and believes the authentication and authorization information provided by the central security entity. The video stream resource is still paranoid, but now when it defers to the local security entity it will grant access to any entity that is authorized by the central security entity.

[0133] Note that the authentication information passed between the video stream entity and the local security entity, and then the local entity and the central security
entity is information specific to the security service and is not part of an identifier exchange between the two entities. This is a subtle and important point, and shows why the local security entity can be considered to be trusting even though it may ultimately reject access to an entity trying to view the camera video stream.

[0134] Figure 15 shows a network with a paranoid security entity. In this example, the central security entity is paranoid to new entities. A configuration such as this prevents rogue devices from entering the network and manipulating the network configuration without the knowledge of the owner of the network. When a new device is added, the central security entity waits until a user manually enters a valid passcode to give the device permission to access the self-configuring network configuration functionality.

[0135] In a network like this, it would be possible to define trusted and untrusted zones. For example, the security entity could be configured to trust all entities on the wired segment of the LAN and to be paranoid towards entities on wireless segments of the LAN.

[0136] Figure 16 shows a network wherein the camera is paranoid to the security entity. The usefulness of this configuration may not be immediately apparent, but it has exciting possibilities. Imagine a mobile user with a laptop that has several network entities on it—for example, video conferencing software, a web camera, and a network file access server. When this user joins a remote self-configuring network network, say at his hotel, he does not want his network services to be available to whomever the hotel’s network deems appropriate. The local security entity on the laptop would find the hotel’s central security entity, but would chose not to trust it and hence would not believe the authorization information it provides. It would then either use a local cache of security information to grant access to the resources on the laptop, or it could optionally seek out a trusted security entity (e.g., at the user’s home network) to provide authentication and authorization.

[0137] Furthermore, the central security entity could be configured to be trusting. In this case, the mobile user’s laptop would have full access to the network configuration functionality of the hotel’s network—allowing his network applications to function properly—without compromising the security of the resources on the laptop itself.
Construction of a Self-Configuring Network

[0138] In the beginning, a Topology Manager is the only element in the network, as shown in Figure 17. Since there are no other elements, the topology is empty. In the next step, a Gateway is added to the network, as shown in Figure 18. It announces itself to the Topology Manager, but does not immediately allocate a subnet. The topology is updated to reflect the presence of the Gateway. The routing information in the Gateway is empty because there are no subnets on the network.

[0139] At this point, the Local IP Address Manager associated with the new Gateway requests a subnet from the Topology Manager. The Topology Manager signals the Gateway that the topology has changed, and the Gateway updates its routing information to reflect the new network topology. Figure 19 shows this configuration.

[0140] Adding new Gateways continues in this fashion. Figure 20 shows the result of adding a new Gateway that is separate from the first, and Figure 21 shows the network after a Gateway is added to subnet 1.

System Configuration

[0141] The system design discussed here allows one to consider real-world configurations that can be used to build a self-configuring network system. In particular, the dependencies in the kernel enforce some restrictions on the physical location of the system elements that comprise it.

[0142] The simplest configuration is to build a single physical device with all of the system elements identified here. Benefits of this configuration include:

- All elements can be configured to be trusting of one another, making the bootstrap process straightforward.
- Any new device added to the network always has a fully functional self-configuring network system available (assuming the primary self-configuring network device is functional).
[0143] The downside to this configuration is that it reduces flexibility and may not be compatible with some network configurations. It is also monolithic in nature, which can be undesirable in some circumstances—in particular, it represents a single point of failure for a self-configuring network system.

[0144] Building the kernel onto a single device provides the infrastructure for devices to establish basic, secure communication amongst one another. Devices that provide Resource Access Management can be added in an arbitrary fashion and their dependencies on the kernel can be resolved securely using network interfaces.

[0145] Distributing the kernel functionality offers a challenge because the kernel offers the network connectivity services required for distribution. In particular, the dependency between the LiPM and the Topology Manager must be resolved. Approaches to solving this problem include:

- Combining a primary LiPM and the Topology Manager so that new devices can be added and establish connectivity dynamically.
- Statically configuring the connectivity of the LiPM and Topology Manager so that they can intercommunicate.
- Using a zeroconf-style protocol to establish rudimentary connectivity for the LiPM and Topology Manager.

[0146] Once the connectivity issues are resolved, Gateways and a CSAM can be added to the network to complete the kernel requirements. As in the previous discussions, an established kernel allows the rest of the self-configuring network to be built.

[0147] As will be apparent to those skilled in the art in the light of the foregoing disclosure, many alterations and modifications are possible in the practice of this invention without departing from the spirit or scope thereof. Generalizations of the foregoing specific implementations are part of this invention. Accordingly, the scope of the invention is to be construed in accordance with the substance defined by the following claims.
WHAT IS CLAIMED IS:

1. A self-configuring network comprising:
   (a) a plurality of devices, each device having a self-configuring network peer and at least one service thereon;
   (b) a plurality of configuration objects; and,
   (c) at least one transmission protocol,

wherein each self-configuring network peer is operable to receive said configuration objects by means of said transmission protocol and provide said configuration objects to said at least one service in response to requests for said configuration objects by said at least one service.

2. The network of claim 1 comprising a topology manager operable to allocate a subnet to a gateway.

3. The network of claim 1 comprising a gateway operable to link subnets and update routing information for the network.

4. The network of claim 1 comprising a local IP address manager configured to manage a pool of IP addresses available on the network.

5. The network of claim 1 comprising a WAN address manager operable to control WAN addresses and routing on the network.

6. The network of claim 1 comprising a traffic manager operable to accept a request from one of said plurality of devices and select a routing mechanism for said request.

7. The network of claim 1 comprising a DNS server operable to resolve LAN names and WAN names into IP addresses.

8. The network of claim 1 comprising a LAN name manager operable to register at least one name for resolution on the LAN.

9. The network of claim 1 comprising a WAN name manager operable to register at least one name, said at least one name usable by WAN devices to access LAN network resources.
10. The network of claim 1 comprising a central secure access manager operable to coordinate authentication and authorization for the network.

11. The network of claim 1 wherein at least one of said plurality of devices comprises a local secure access manager operable to coordinate authentication and authorization for said at least one of said plurality of devices.

12. The network of claim 1 comprising a network monitor operable to allow a user to monitor a state of the network and to adjust a network configuration.

13. A device for use with a self-configuring network, said device comprising:
   (a) at least one service, each service having at least one required configuration object;
   (b) a service interface operable to translate information relating to said at least one required configuration object in a predetermined format;
   (c) a self-configuring network core operable to receive said information in said predetermined format and generate a request for said at least one required configuration object; and,
   (d) a protocol interface operable to translate said request into a network protocol,
wherein said self-configuring network core is operable to obtain said at least one required configuration object from said self-configuring network and provides said at least one required configuration object to said at least one service.

14. The device of claim 13 further comprising at least one available configuration object, wherein said self-configuring network core is operable to generate an offer of at least one available configuration object and said protocol interface is operable to broadcast said offer over said self-configuring network using said network protocol.

15. A method of configuring a device on a self-configuring network, said device having at least one service thereon, the method comprising, for each service:
   (a) determining at least one required configuration object by means of a software component running on said device;
(b) broadcasting a request for said at least one required configuration object in a network protocol by means of a protocol interface;

(c) receiving said at least one required configuration object from said self-configuring network by means of said protocol interface; and,

(d) providing said service with said at least one required configuration object by means of said software component.

16. A method for routing data packets of an application across a network using a multi-layered protocol, the method comprising associating integrity constraints for one or more of the layers and preserving layers associated with the integrity constraints while transporting the data packets across the network.

17. The method of claim 16 comprising associating access constraints with the layers.

18. A network comprising any new and inventive combination or subcombination of elements described herein.

19. A network connected device comprising any new and inventive combination or subcombination of features described herein.

20. A method of operating a network, the method comprising any new and inventive combination or subcombination of steps described herein.

21. A method of configuring a network, the method comprising any new and inventive combination or subcombination of steps described herein.
Figure 8

<table>
<thead>
<tr>
<th>Routing Key</th>
<th>MAC</th>
<th>IP</th>
<th>Transport</th>
<th>Application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Integrity</td>
<td>MAC</td>
<td>IP</td>
<td>Transport</td>
<td>Application</td>
</tr>
<tr>
<td></td>
<td>IP</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Transport</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Application</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Figure 9

ADN Device

- Connectivity configuration
- LIPM: Connectivity
- Network security
- CSAM: Security
- Resource access
- Traffic Manager: Resource Access
- DNS Server: Resource Access
- Resource name lookup

Figure 11
Update network configuration via configuration functionality

Gateway
Trusts network entities

Camera
Paranoid to video stream access

Trust

Attempt to access video resource from external source: subject to A & A by camera

Figure 13

Update network configuration via configuration functionality

Gateway
Trusts network entities

Camera
Paranoid to video stream access

Security Entity
Trusts network entities

Checks A & A for video stream access

Figure 14
Figure 15

1. Registers itself for use of network ADN resources
2. Authenticates video stream access
Figure 16

1. Registers itself for use of network ADN resources
2. Authenticates video stream access IF trusting relationship has been established. May alternately provide its own A & A or seek a trusted security entity.
Figure 21