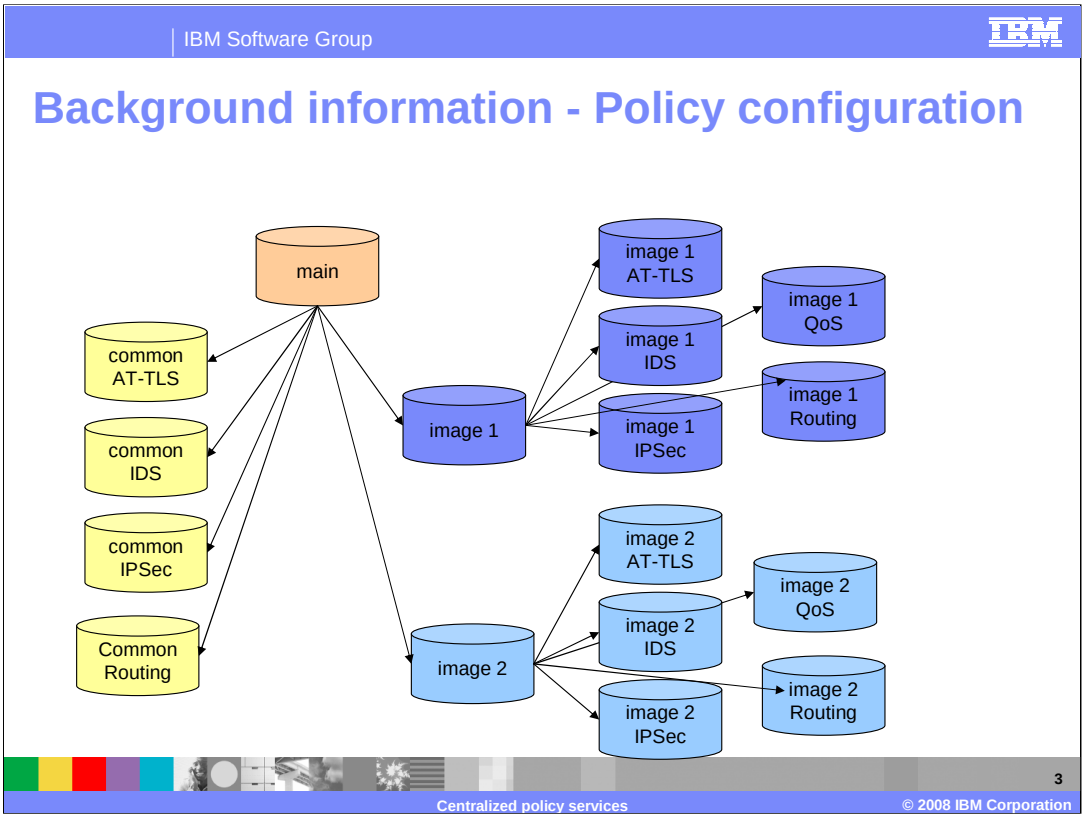
This presentation discusses the Centralized Policy Services function for the z/OS V1R9 Communications Server.

Background information - Policy agent

- The policy agent manages policy definitions for a variety of uses
  - **Quality of service** (assign priority, limit connections, measure QoS characteristics, and so on)
  - **Intrusion detection services** (regulate traffic, decide how to report intrusions, and so on)
  - **IP security** (filter traffic, define key exchange data for the IKE daemon)
  - **Application transparent TLS** (provide TLS/SSL services on behalf of applications)
  - **Policy-based routing** (allows a route to be selected based on multiple criteria)
- The policy agent is one element of a set of components that provide policy based networking

First, a quick refresher on the major attributes of the policy agent. Primarily, the policy agent is a manager for sets of policy definitions. The policy definitions are categorized into different policy types, as shown on this slide. Each policy type can be used to accomplish various goals, some of which are shown in parentheses for each policy type.

The policy agent is one of several components that provide a more general function known as *policy based networking*. Policy based networking is a way of accomplishing a set of network goals through the use of policy definitions. For example, one network goal may be to provide better quality of service (QoS) for one set of traffic as compared to another set. Policies can be defined to set the IPv4 type of service (TOS) or IPv6 traffic class for the two sets of traffic, to assist in obtaining the required QoS from the network.



This slide graphically depicts the entire set of configuration files that can be used to define the different policy types and general policy agent configuration. It is important to understand that various subsets of the configuration files shown might be used, depending on the different policy types in use and the number of TCP/IP stacks supported by an instance of the policy agent. Also note that Lightweight Directory Access Protocol (LDAP) configuration is not shown in this diagram.

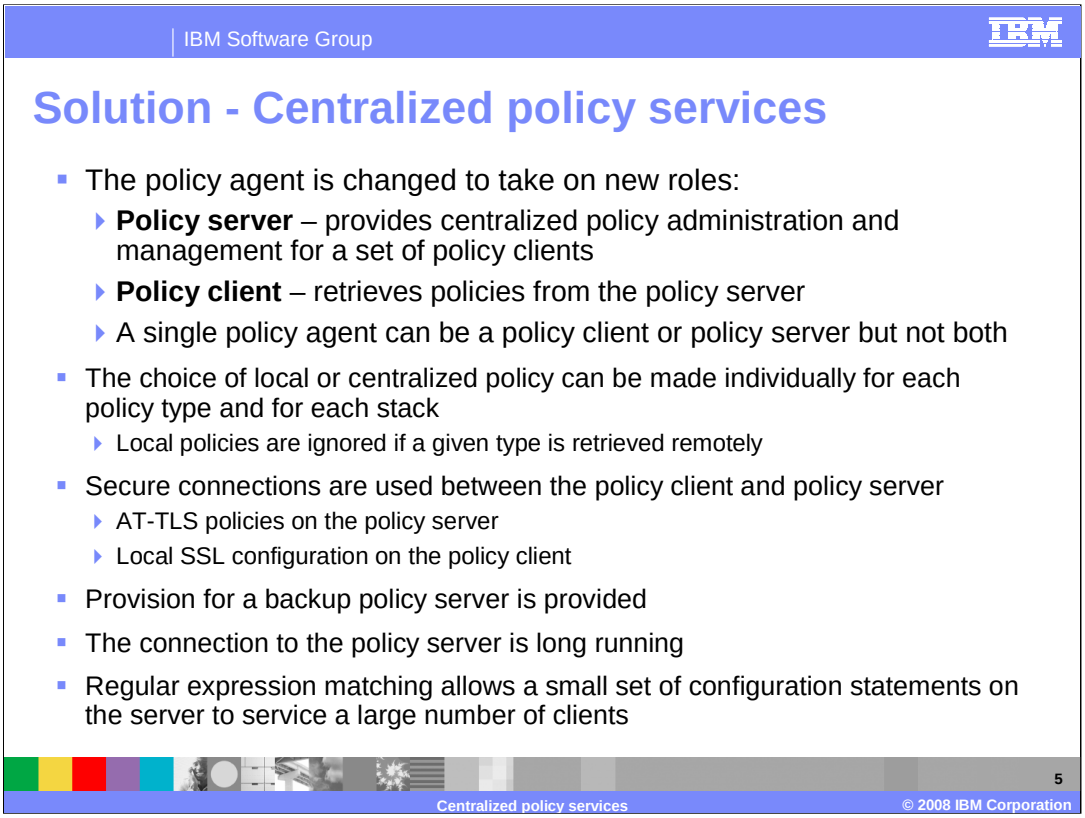
When the policy agent is started the main configuration file is identified using a standard search order. This file in turn can point to one or more image configuration files using the TcpImage statement. Each image configuration file is used to configure policies for one TCP/IP stack. The image files can in turn point to image-specific files for the different policy types. The main configuration file can point to common files for all policy types except QoS. A given common configuration file applies to all TCP/IP stacks. This allows policy definitions that are not unique for each TCP/IP stack to be placed in the common file, and those that are unique to be placed in each image-specific file.

The image QoS file is optional – QoS definitions can be placed directly in the image configuration file instead of a separate file. Also, the statements in the image configuration files can instead be placed directly in the main configuration file, by specifying a TcpImage statement without a separate image file path name. However, such definitions will be shared by all TCP/IP stacks that don't have their own separate image configuration file.

Problem statement - Policy management

- The scope of policy agent policies continues to widen, with new policy types added over the last several releases
- Local management of policies is therefore becoming a larger administrative burden
- Using LDAP as a centralized policy repository is not possible (LDAP only supports QoS and IDS)

The problem being solved by centralized policy services is primarily one of policy management. Each of the last several releases has introduced a new policy type, and the policy agent configuration shown on the previous slide needs to be replicated on each system. If the IBM Configuration Assistant for z/OS Communications Server is used to configure policy definitions, it also must be replicated on (or have connectivity to) each system. It isn't possible to use LDAP as a centralized policy repository, because the LDAP implementation only supports the QoS and IDS policy types.

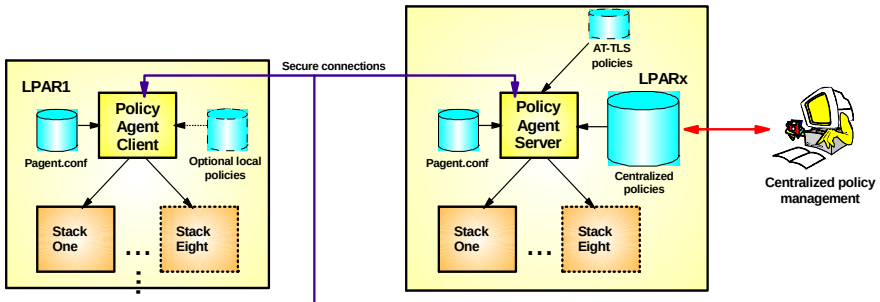


The solution is to use the policy agent itself as a centralized policy repository. The new roles of *policy server* and *policy client* are introduced. The policy server provides centralized administration and management of policy definitions. The IBM Configuration Assistant only needs connectivity to the policy server, if no local policies are defined on the policy clients.

This slide identifies the major characteristics of centralized policy services. Note that the local or remote policies can be used for each policy type, and for each TCP/IP stack, individually. These choices can be changed at any time without restarting the policy agent. This allows a gradual and controlled migration from local to remote policies.

Also note that secure long running connections are used between the policy clients and the policy server. The policy server uses local AT-TLS policies to accomplish this. But it isn't possible to use AT-TLS policies on the policy client, because of the chicken-and-egg problem: if the AT-TLS policies reside on the policy server, the policy client would need to connect to the policy server in order to obtain the policies that secure that very connection. For this reason, the policy clients are configured as SSL clients, using local definitions in the image configuration files.

Centralized policy services



- Centralized policy management and storage for a cluster of nodes that use a common networking policy infrastructure based on the policy agent technology
- Centralized management becomes increasingly important as networking policy scope widens (QoS, IDS, IPsec, AT-TLS, PBR, and so on.)
- Some of the policy types can benefit from additional centralized services, for which the centralized policy services will serve as a base

This picture shows an overview of the centralized policy services solution. On the left side are several policy clients. Each policy client can use local configuration file as usual, if needed. On the right side is the policy server. Centralized policies are defined, but are not installed in any TCP/IP stacks, on the policy server. These centralized policies are retrieved by the policy clients using the existing policy agent API (PAPI).

The IBM Configuration Assistant can be used to define the centralized policies and local policies for the policy server and policy client (this is not shown).

To take full advantage of this solution, local policies should not be defined on the policy clients. The policy server is not itself considered a policy client, so local policies on the policy server are normal and expected.

The problem being solved by centralized policy services is primarily one of policy management. Each of the last several releases has introduced a new policy type, and the policy agent configuration needs to be replicated on each system. If the IBM Configuration Assistant for z/OS Communications Server is used to configure policy definitions, it also must be replicated on (or have connectivity to) each system.

Centralized policy services provides a centralized policy management and storage for a cluster of nodes that use a common networking policy infrastructure based on the policy agent technology. Initially a cluster of z/OS nodes is supported. However it can be extended to act as centralized networking policy server for heterogeneous nodes. Centralized management becomes increasingly important as networking policy scope widens (QoS, IDS, IPSec, AT-TLS, PBR, and so on.).

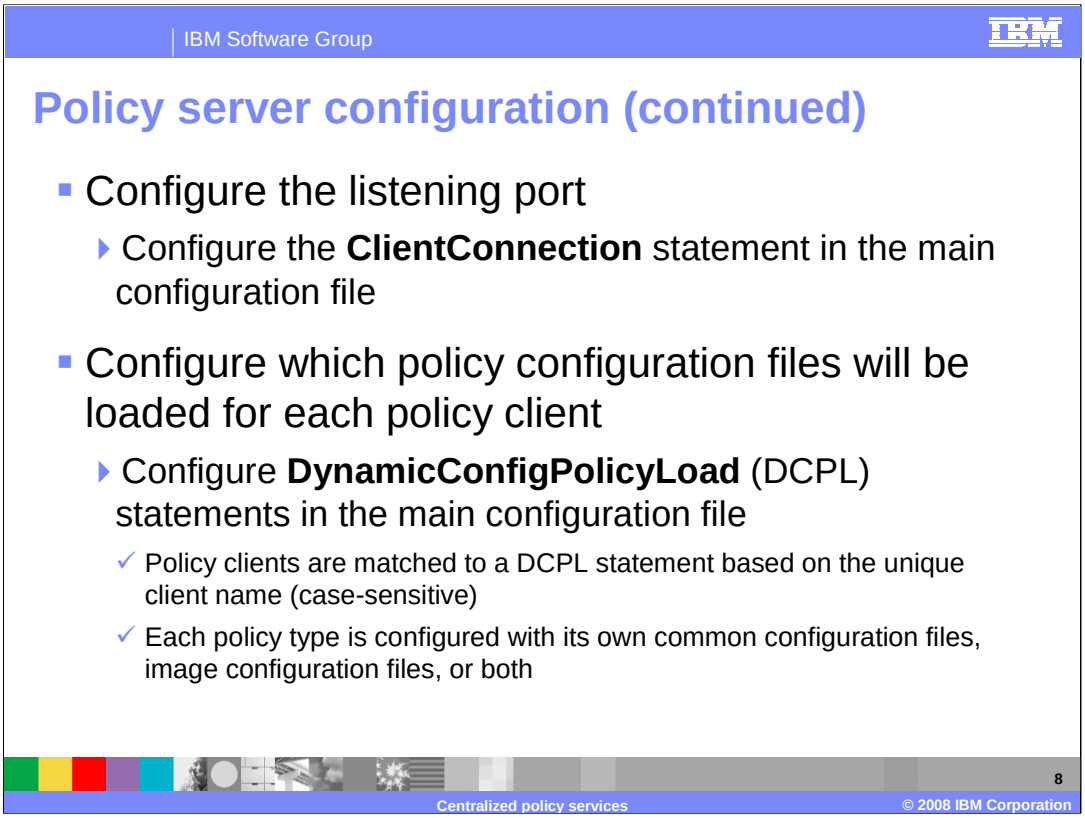
The policy agent is changed to take on new roles policy server and policy client. The policy server provides centralized policy administration and management for a set of policy clients. The policy client retrieves policies from the policy server. A single policy agent can be a policy client or policy server but not both.

Policy server configuration

- Configure appropriate security mechanisms to allow policy client connections
  - Configure SERVAUTH profiles to permit policy clients to retrieve policies
  - Configure a set of user IDs for policy clients
    - ✓ Used to authenticate policy clients
    - ✓ Used to access SERVAUTH profiles
      - EZB.PAGENT.*sysname.image.ptype*
  - Permit PAGENT to the BPX.DAEMON FACILITY class profile
    - ✓ Optionally configure program control for PAGENT
  - Configure PTKTDATA class profiles if any policy clients use PassTicket authentication
  - Configure AT-TLS policies to allow secure connections from policy clients
    - ✓ Permit PAGENT to the EZB.INITSTACK.*sysname.tcpprocname* SERVAUTH profile

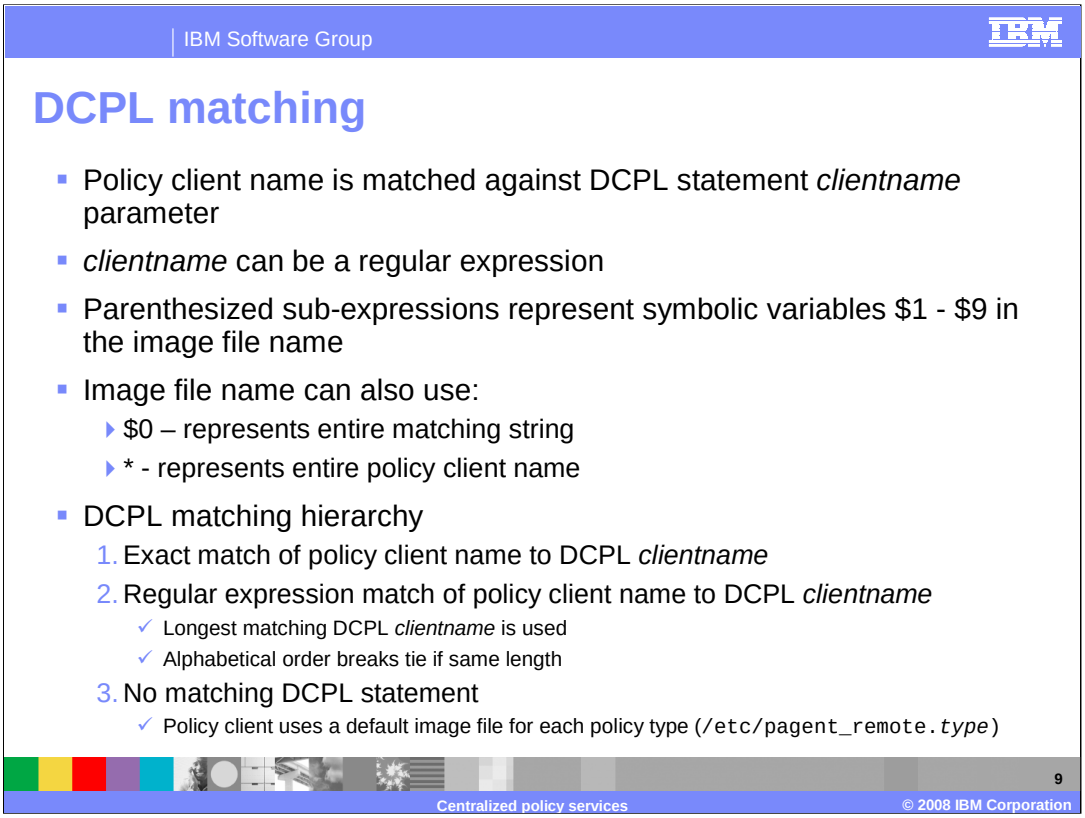
The first step is configuring various security mechanisms to allow policy clients to connect to the policy server. The first item deals with the existing security product EZB.PAGENT.*sysname.image.ptype* SERVAUTH profile. This profile is currently used to authorize policy agent clients (such as the pasearch command and the IKE daemon) to access various policy types for different TCP/IP stacks. For z/OS V1R9 Communications Server, the *image* portion of the profile name is now generic, and can be either a TCP/IP stack or a policy client name. This profile must exist to allow policy clients to retrieve policies. Each policy client presents a user ID when it connects to the policy agent. This user ID is used to authenticate with the policy server (using either a password or PassTicket), and to access the EZB.PAGENT.*sysname.image.ptype* SERVAUTH profiles defined in the previous step. A unique user ID can be created for each policy client, but that is not a requirement. For example, you might decide to use one user ID for a system or other set of policy clients.

You can optionally use program control for the policy agent. This provides enhanced control for who is allowed to run the policy agent. To do this, permit the policy agent user ID to the BPX.DAEMON FACILITY class profile. You may want to use PassTickets instead of passwords to authenticate policy clients with the policy server. This prevents the passwords from being coded in the policy client image configuration files. However, the use of PassTickets requires that you define PTKTDATA class profiles on both the policy client and policy server. These profiles contain a secure key that is used to generate the one-time usage PassTickets. The profile name for these PTKTDATA profiles must be PAGENT on the policy clients, but can be either PAGENT or PAGENT.*userid* on the policy server. The time of day clocks on the policy server and all policy clients using PassTickets must also be reasonably synchronized (within a few minutes). This is because PassTickets are only valid for 10 minutes between generation (on the policy client) and verification (on the policy server). You must configure AT-TLS policies on the policy server to allow secure connections to be established from policy clients. These policies must point to a key ring that contains the appropriate server certificate. Because the policy agent uses AT-TLS policies, it must be permitted to the EZB.INITSTACK.*sysname.tcpprocname* SERVAUTH profile. This allows the policy agent to establish sockets before the AT-TLS policies are installed.



The next step to configure the policy server is to define a listening socket using the ClientConnection statement. The only parameter on this statement is the port number. The default port number is 16310. The policy agent listens for connections using IN6ADDR_ANY and this port number. You should reserve this port using the PORT statement in the TCP/IP profile.

The last policy server configuration step is to define one or more DynamicConfigPolicyLoad statements. These statements determine what configuration files are used to contain the centralized polices for all policy clients. When a policy client connects, an attempt is made to match the case-sensitive client name to the *clientname* parameter on a DCPL statement. Default values are used if a DCPL statement can't be matched. The *clientname* parameter can use regular expression characters to match a set of policy clients. Each DCPL statement points to a common configuration file and image-specific configuration files for each policy type. The image-specific configuration file names can contain symbolic replacement or wildcard variables, so that the resulting configuration file is unique for each policy client that matches the DCPL statement. Dynamic update using the -i startup option is not supported for these files.



As noted on the previous slide, policy clients are matched to a DCPL statement using the *clientname* parameter, which can be a regular expression. This regular expression is similar to (but not exactly like) regular expressions used on UNIX commands like grep and awk. Parentheses can be used in the regular expression to create sub-expressions. These sub-expressions can then be represented by symbolic replacement variables in the image configuration file names. Stay tuned for an example to help make this clearer.

Other symbolic replacement variables can also be used:

$0 represents the entire portion of the policy client name that matches the regular expression (in other words it isn't limited to a sub-expression).

* is a wildcard that represents the entire policy client name.

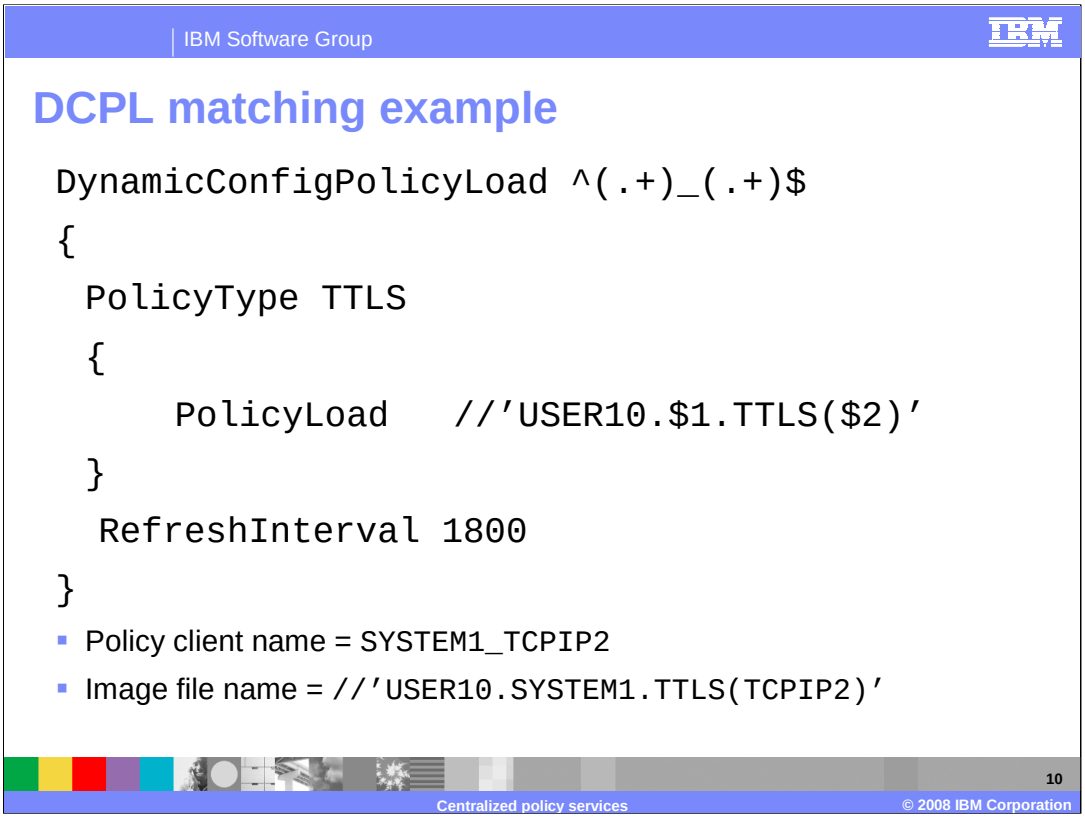
Note that for some regular expressions, $0 and * might resolve to the same value, while for others they will not. If you really want to substitute the entire policy client name, use the * wildcard.

A given policy client is matched to a DCPL statement using a matching hierarchy. The order that the DCPL statements are specified in the configuration file is not important.

1.An exact match between the policy client name and the DCPL statement. In this case the DCPL *clientname* does not contain any regular expression characters (it is just a string, like "client42"). You could use this form to override a more general DCPL statement for a specific policy client, for example.

2.A regular expression match to a DCPL statement. If multiple statements could match, the one with the longest *clientname* parameter is chosen. If multiple matching statements exist with the same length *clientname*, the one chosen is based on alphabetical order of the *clientname*.

3.If no DCPL statement matches, default values are used for the configuration file names and other parameters. The default configuration file names take the form: `/etc/pagent_remote.type`, where *type* is one of the following: `ids, ipsec, qos, routing, ttls.`



Putting the pieces together, this slide shows a DCPL statement with the *clientname* "^(.+)_(.+)$". The ^ (caret) and $ (currency) characters delineate the name. These characters represent the start and end of a string, so using them means the entire policy client name, not just a portion, must match. The regular expression contains 2 sub-expressions contained in parentheses, separated by an underscore. The underscore is not a regular expression character, so it literally matches an underscore in the policy client name. Each sub-expression matches one or more of any character.

The image configuration file name contains 2 symbolic substitution variables, $1 and $2, that correspond to the sub-expressions in the regular expression.

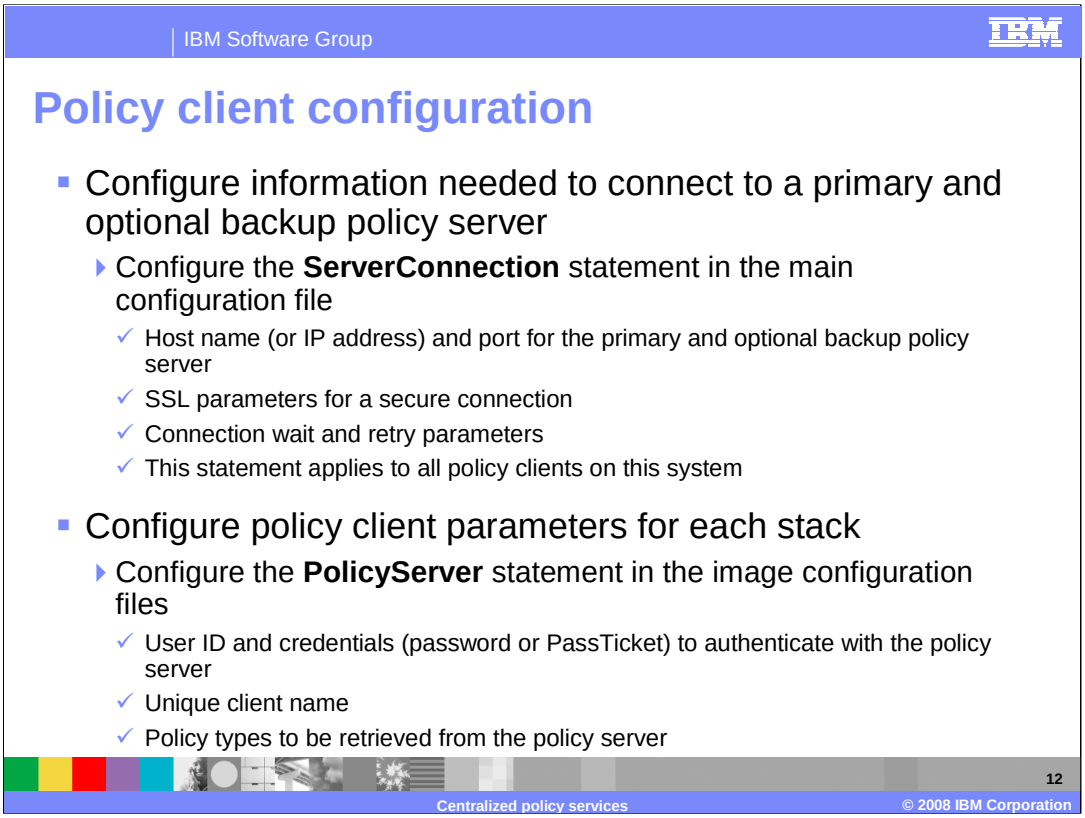
Now, suppose the policy client named "SYSTEM1_TCPIP2" connects to the policy server. Since this name consists of 2 strings separated by an underscore, it matches the regular expression on the DCPL statement. The portions of the policy client name that match the 2 sub-expressions replace the symbolic substitution variables, so the resulting image file name is "//'USER10.SYSTEM1.TTLS(TCPIP2)'".

A scheme such as this example allows a PDS to be set up for each policy type for each remote system, containing a member for each policy client on those systems. The power of regular expressions and symbolic replacement variables allows many other schemes to also be used.

DCPL binding

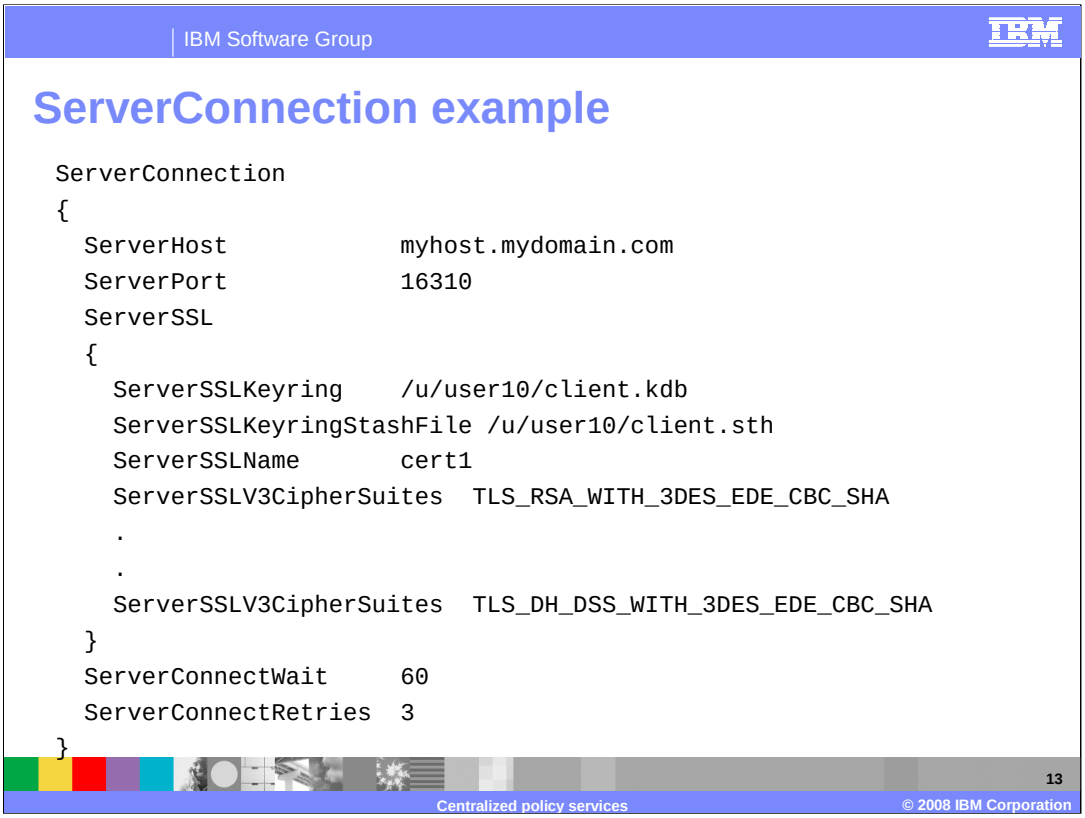
- A DCPL statement is bound to a policy client until:
  - The policy client disconnects
  - The connection to the policy client ends
  - The DCPL statement is removed
    - ✓ All policy clients are bound to a new DCPL statement (or default values)

Once a DCPL statement is matched to a policy client, it is bound to that policy client until one of the events listed on this slide occurs.



The first step in configuring the policy client is to specify the ServerConnection statement in the main configuration file. This statement establishes configuration data that apply to all policy clients on this system. The slide shows the data that can be configured.

For each policy client (TCP/IP stack) on the system, configure a PolicyServer statement in the image configuration file. This statement contains parameters to authenticate the policy client with the policy server, the policy client name, and which policy types to be retrieved. The policy client name must be unique. If a name is not specified then the default of *systemname_stackname* is used. The FLUSH and PURGE parameters can also optionally be configured on the PolicyServer statement.

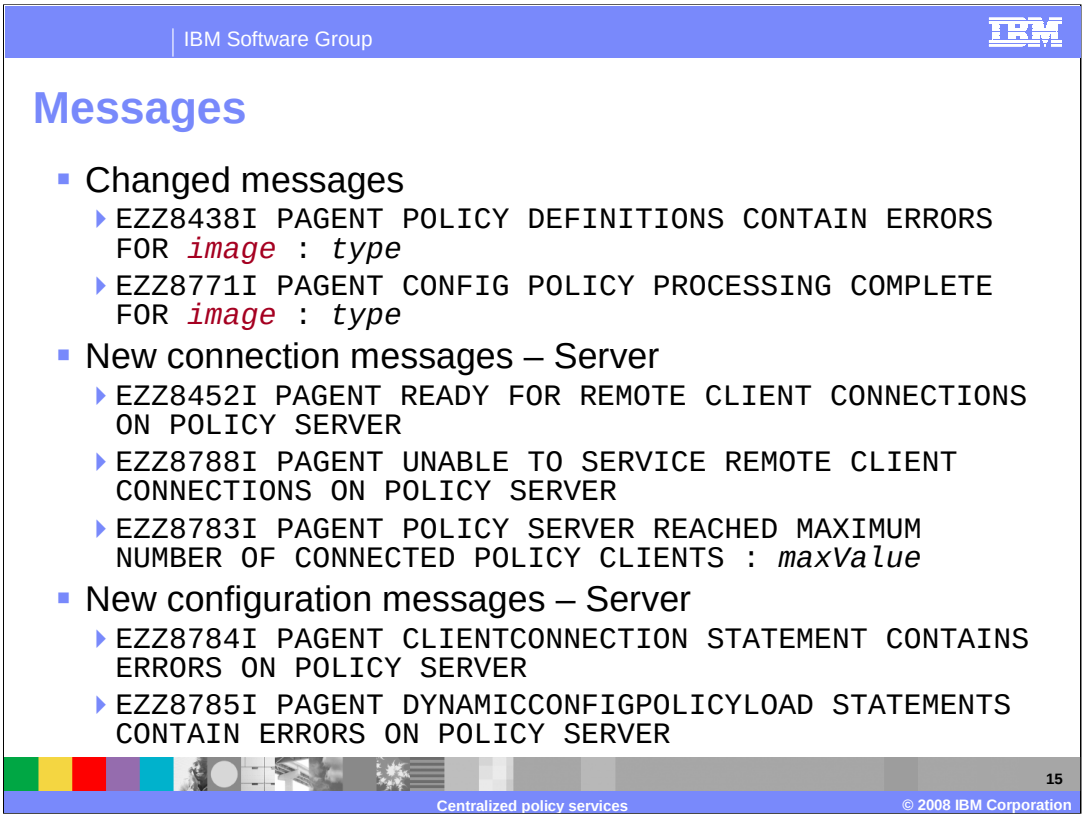


Here's a sample ServerConnection statement. In this example the primary policy server, located at myhost.mydomain.com and listening on port 16310, is configured. A backup policy server is not configured. The policy agent, acting as a policy client will try to connect to the policy server. If the connection attempt fails, then the policy agent will retry 3 times.

PolicyServer example

```
PolicyServer
{
  Userid                USER10
   AuthBy               PassTicket
   ClientName           SYSTEM1_TCPIP2
   PolicyType           IPSec
   PolicyType           TTLS
   {
      FLUSH
      NOPURGE
   }
}
```

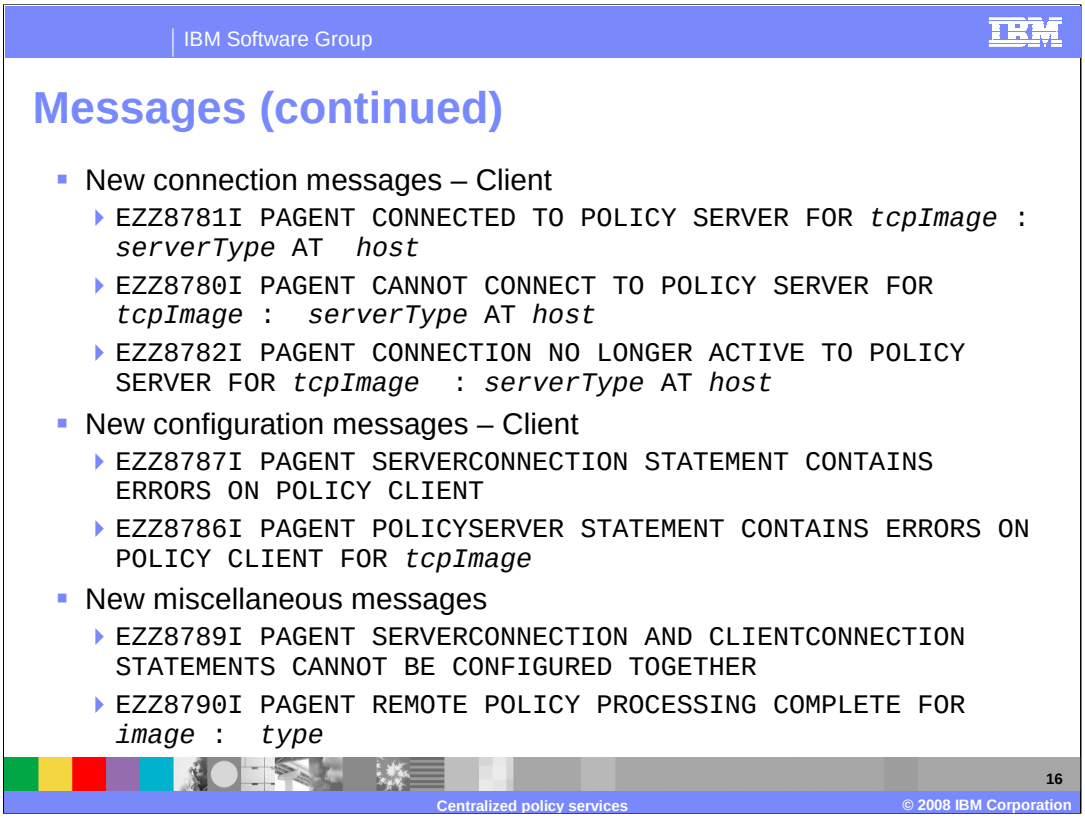

Here's a sample PolicyServer statement. Note that the additional set of braces is optional for the PolicyType parameter. If you don't need to specify the FLUSH/NOFLUSH or PURGE/NOPURGE parameters for a policy type, the braces can be omitted. The FLUSH/NOFLUSH and PURGE/NOPURGE parameters configured on the TcpImage statement are used as defaults.



The configuration chores are complete. Now let's take a look at new and changed messages. Shown here are the only messages that changed. The change is to replace the TCP/IP stack name variable with the more generic *image* variable. The *image* can identify either a TCP/IP stack or a policy client name.

This slide also shows connection-oriented messages that can be issued on the policy server. Note that the EZZ8783I message is issued if more than the maximum number of policy clients try to connect to the policy server, so you are unlikely to see this message. The EZZ8452I message indicates that the ClientConnection statement is properly configured and the policy agent is listening for remote connections. If for any reason the policy agent is unable to listen for remote connections, but the ClientConnection statement is configured – for example no TCP/IP stacks are started – you'll see message EZZ8788I

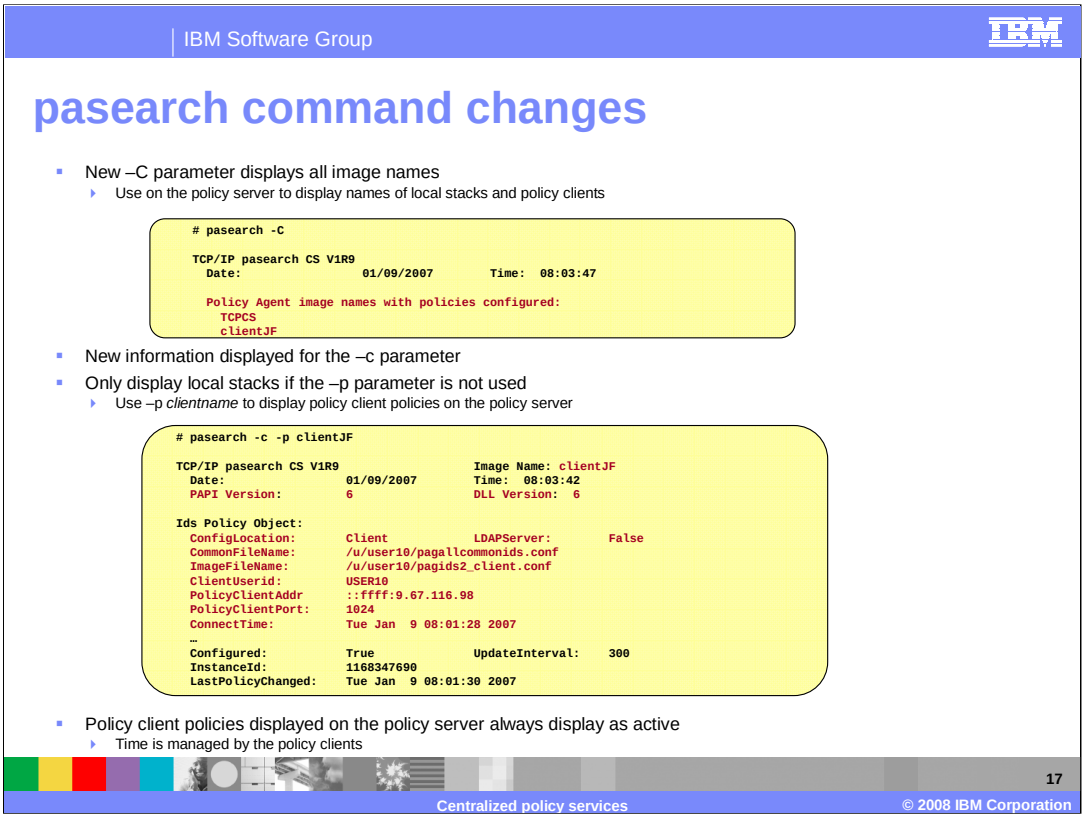
Messages EZZ8784I and EZZ8785I indicate problems with the new configuration statements for the policy server, CLIENTCONNECTION and DYNAMICCONFIGPOLICYLOAD.



As with the policy server, there are some connection-related messages issued on the policy client. The first 2 messages shown indicate success or failure when trying to connect to a policy server. The EZZ8782I message is issued if an active connection to the policy server ends. In most cases, the policy client will retry the connections to the primary and backup policy servers.

Messages EZZ8787I and EZZ8786I indicate problems with the new configuration statements for the policy client, SERVERCONNECTION and POLICYSERVER.

If you try to configure the policy agent as both a policy server and a policy client, you'll see the EZZ8789I message. Message EZZ8790I is a companion to the existing EZZ8771I message. The new message is issued to the client console to distinguish remote policies from local policies (the existing message is issued for local policies). The new message is also issued on the policy server, but only to the log file, to avoid flooding the console if a large number of policy clients exist.



A new –C parameter displays all image names. Here's an example of the output for the new –C parameter. When issued on the policy server, it shows a list of the local TCP/IP stack names, followed by a list of connected policy clients. This example shows one of each. Note that you can also issue this command on the policy client, but it only displays local TCP/IP stack names.

There are several changes to the existing output for the –c parameter. First, note that the header information is slightly changed – it shows **Image Name** instead of **TCP/IP Image**. This is to accommodate policy client names and stack names. This change applies to the output for most parameters, not just the –c parameter. For the –c output, a new line in the header shows the PAPI version negotiated between the policy client and policy server (**PAPI Version**) and the local PAPI version (**DLL Version**). For V1R9 these will always be 6, which is the current PAPI version. In future releases, these could be different if the versions of the policy client and policy server are different. When displaying policy types that use local configuration files, the **ConfigLocation** will show **Local**, and the **LDAPServer** will be **TRUE** or **FALSE** to indicate if LDAP is being used for this type. The common and image file names are also shown.

In the pasearch –c example shown, a policy is being displayed for a policy client on the policy server. When displaying policies for policy clients on the policy server, **ConfigLocation** indicates **Client**. Note that the policy client name is displayed as the image name in the header. In addition to the common and image file names, the following items are also displayed:

•**ClientUserid** – the user ID used by the policy client when it connected

•**PolicyClientAddr** – the IP address of the policy client

•**PolicyClientPort** – the port of the policy client

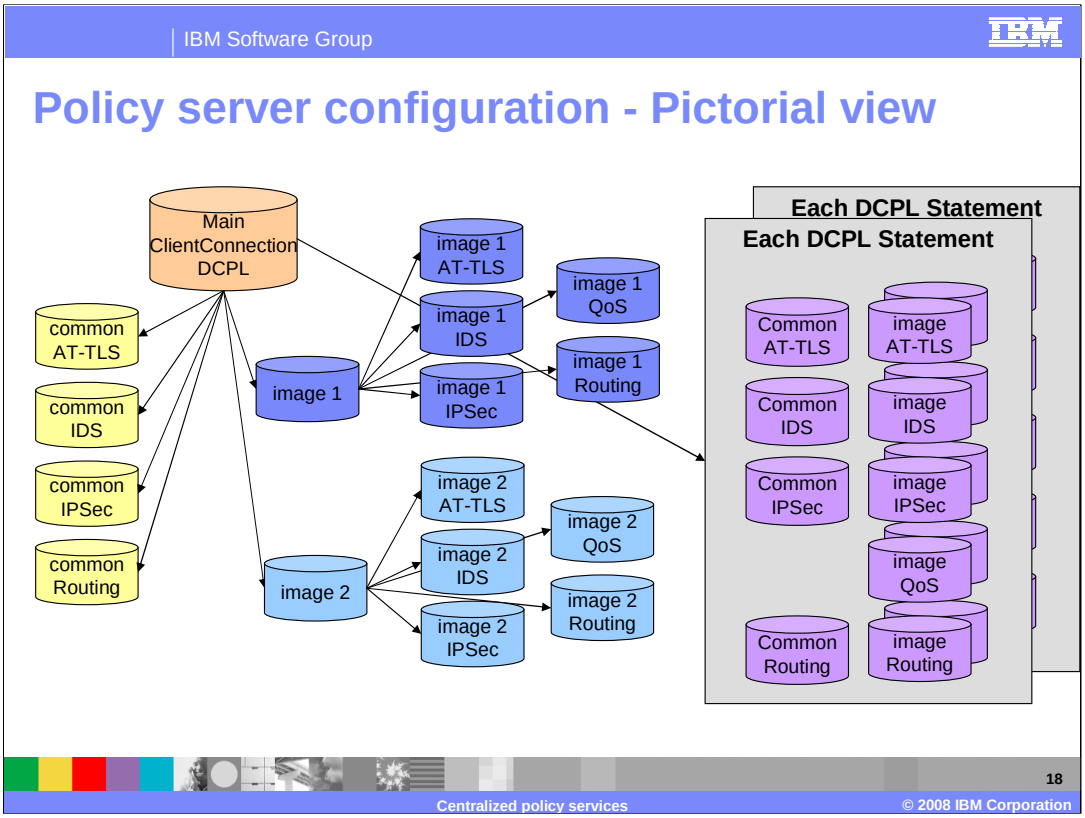
•**ConnectTime** – the date and time when the policy client connected

When displaying policies retrieved remotely on the policy client, **ConfigLocation** indicates **Remote**. The configuration file names are not displayed in this case, because they do not exist locally, and it might not be appropriate for the policy client to know the configuration file names. In addition to the ClientName, ClientUserid and ConnectTime values, the following items are also displayed:

•**PolicyServerAddr** – the IP address of the policy server

•**PolicyServerPort** – the port of the policy server

•**PolicyServSysName** – the system name of the policy server



That concludes the changes to the externals for centralized policy services. Now let's look at some more details of the solution.

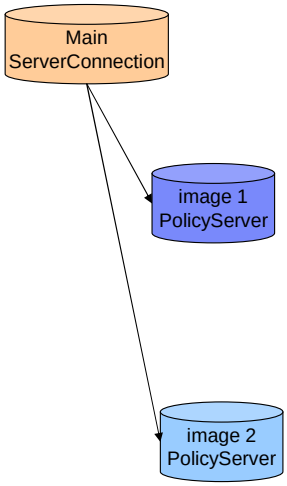
This slide graphically shows the changed configuration on the policy server. The left side of the slide is the same as the earlier slide that showed the configuration before V1R9. All of that configuration can still be used to configure local policies on the policy server.

The ClientConnection statement and one or more DynamicConfigPolicyLoad statements need to be configured in the main configuration file, in order to provide policy services for policy clients.

The gray boxes show the new configuration files needed to define centralized policies. Each gray box represents one DCPL statement, which in turn provides configuration for a set of policy clients. There is one common configuration file for all policy types except QoS. There are one or more image configuration files for each policy type. If the image file names on the DCPL statement use symbolic replacement or wildcard variables, then each policy client using a given DCPL statement needs a separate image configuration file.

More configuration is required, but is centralized on one system. Only one instance of the IBM Configuration Assistant is needed (if used), and it only needs to connect to the policy server.

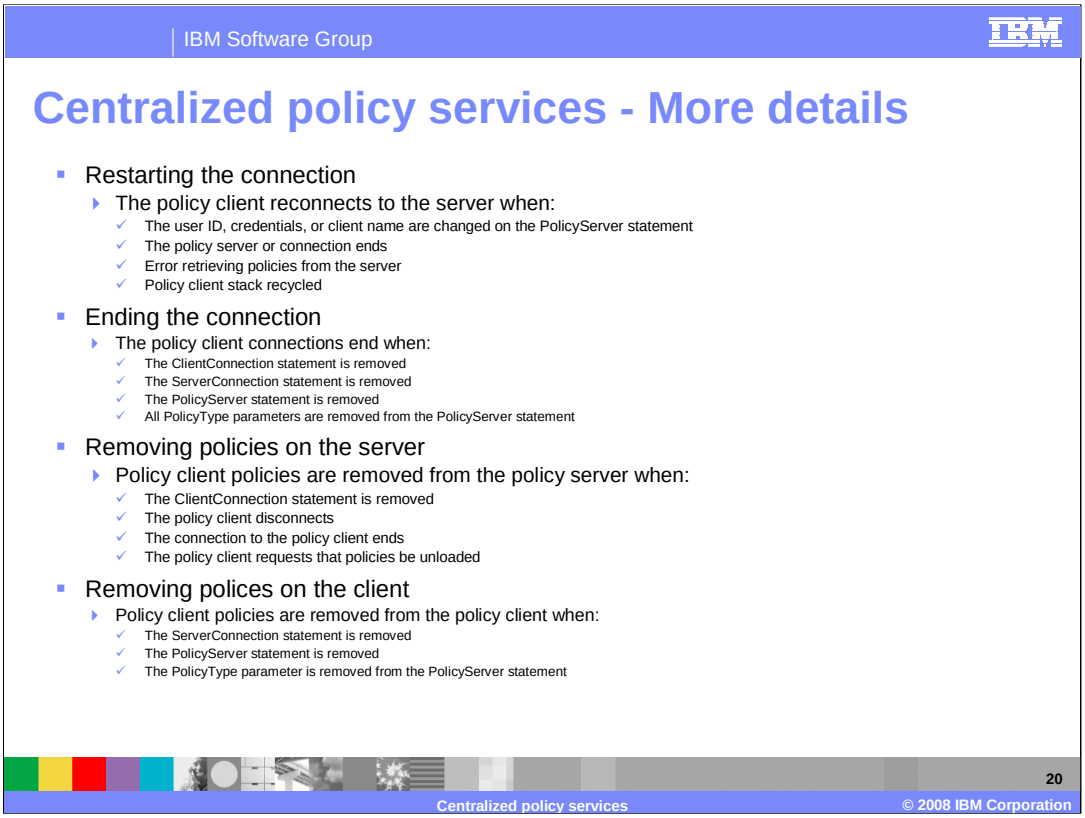
Policy client configuration - Pictorial view



This slide shows the changed configuration on the policy client, if all policy types are retrieved from the policy server. If some local policies are still needed, some subset of the left side of the previous slide will still be needed.

All that is needed if all policy types are retrieved remotely is the main configuration file and an image configuration file for each TCP/IP stack.

The ServerConnection statement must be configured in the main configuration file, and the PolicyServer statement must be configured in each image configuration file, in order to retrieve policies from the policy server.

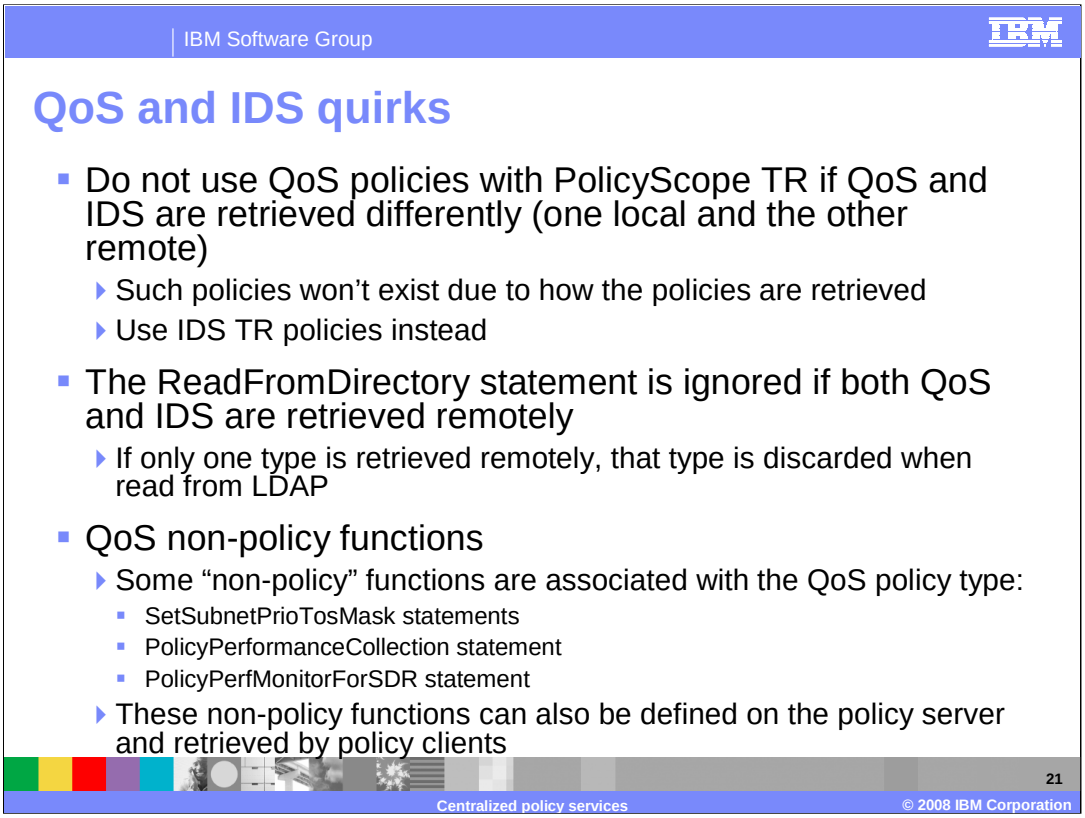


As noted earlier in the presentation, the connections between policy clients and the policy server are long running. This slide shows conditions that result in the connection being restarted.

Some conditions result in the connection ending without being restarted. These are configuration changes that basically remove the policy client or policy server role from the policy agent. When the ClientConnection statement is removed from the policy server main configuration file, then the connections for all policy clients are ended. When the ServerConnection statement is removed from the policy client main configuration file then all connections from that policy client to the policy server are ended.

Policies that have been loaded into the policy agent for policy clients are removed from the policy agent, acting as the policy server, for the conditions shown on this slide. A policy client can request that polices be unloaded by removing a PolicyType from the PolicyServer statement.

Policies that have been retrieved from the policy server are removed from the policy agent and the TCP/IP stack for the conditions shown on this slide. If local policies are configured for any of the affected policy types, they are then loaded. The ServerConnection statement, in the policy client main configuration file, being removed affects all TCP/IP stacks and policy types. The PolicyServer statement, in the policy client image configuration file, being removed affects all policy types for the associated TCP/IP stack. The PolicyType parameter being removed from the PolicyServer statement affects only the removed policy types for the associated TCP/IP stack.



There are a few exceptions to the rule. For QoS and IDS policies there are two things of interest:

1) QoS policies that use PolicyScope TR should not be specified if QoS and IDS policies are retrieved differently (one local and the other remote). Due to how such policies are transformed into IDS TR policies, they will not exist after all policies have been loaded. If you want to use the TR function, configure IDS TR policies instead.

2) Because policies loaded from an LDAP server are considered local, if both QoS and IDS policies are retrieved remotely, the ReadFromDirectory statement is ignored, and the LDAP server is not even contacted. Note that if only one of QoS or IDS policies are retrieved remotely, policies of that same type may be read from the LDAP server, but are discarded.

QoS also includes some functions that are not strictly policies. These "non-policy" functions are shown on this slide. They can be configured on the policy client just like QoS policies, and retrieved by policy clients.

Diagnosis - Documentation for problems

- Gather documentation from both the policy client and policy server
- Policy agent logs are the primary documentation

- Both LogLevel and debug level are provided
  - Minimum LogLevel 127 – use 511 if possible (but this uses more log space)

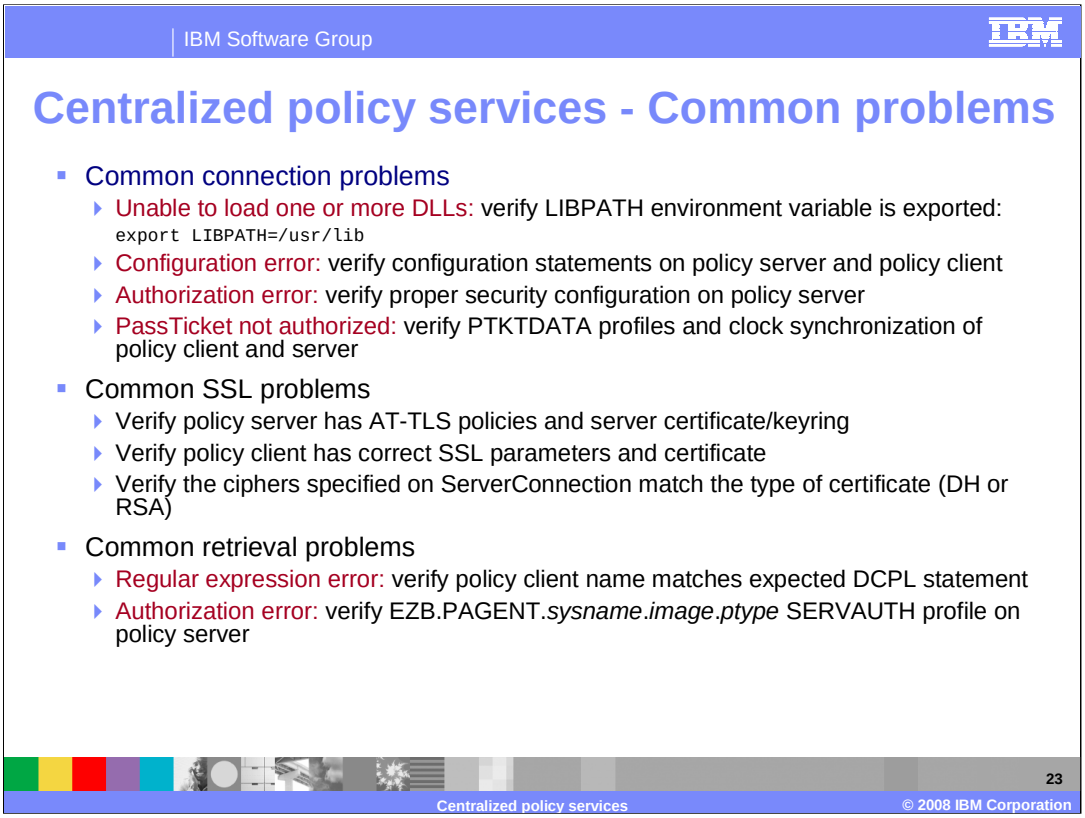
    - Log space can be increased using:

```
export PAGENT_LOG_FILE_CONTROL=kbytes,number
```

  - Use debug level 128 for connection or remote policy retrieval problems

Gather documentation from both the policy client and policy server if you encounter problems with centralized policy services. The primary documentation required is the set of policy agent log files. For successful debugging, LogLevel 127 should be considered the minimum.

See IP Diagnosis for more complete information on gathering documentation.



This slide shows some common connection problems that you might encounter. See the z/OS V1R9 Communications Server IP Diagnosis Guide for more details. For the PassTicket not authorized problem see z/OS Security Server RACF Security Administrator's Guide, Single sign-on function.

The connections use SSL, and here are some of the common SSL problems that you might encounter. A common error is incorrect SSL configuration for the policy client. If the server AT-TLS policy uses HandshakeRole Server, ServerConnection ServerSSLName parameter must specify the server's certificate. If server AT-TLS policy uses HandshakeRole ServerWithClientAuth, ServerConnection ServerSSLName parameter must specify the client's certificate. See the z/OS V1R9 Communications Server IP Diagnosis Guide for more details. Also see the z/OS V1R9 Communications Server IP Configuration Guide, Appendix B. TLS/SSL Security for detailed information on certificates and establishing a correct SSL environment.

There are some common policy retrieval problems that you might encounter. The policy server logs a message for a successful match of a client name to a DCPL statement in addition to an unsuccessful match. If a regular expression error is encountered then view the messages in the policy server's log file. Following are examples of a log messages:

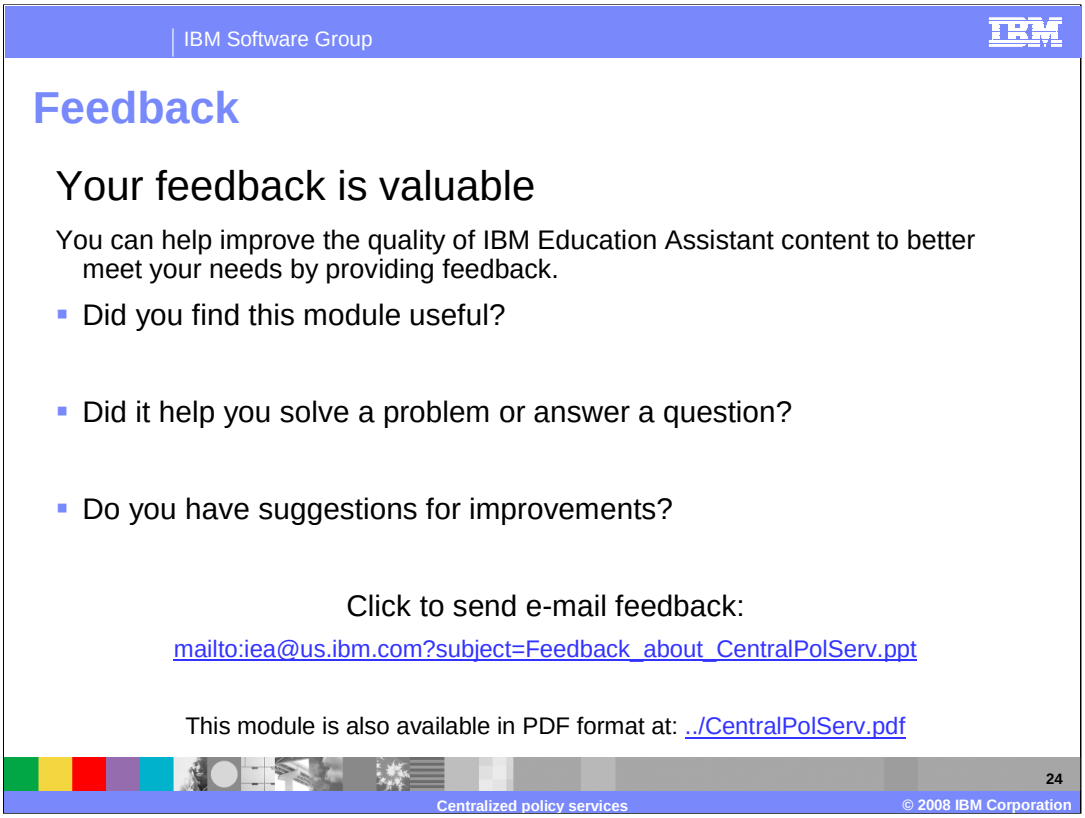
Log message for successful match:

```
client PEP 'clientJF' using DynamicConfigPolicyLoad statement
'client(.*)'
```

Log message for unsuccessful match:

```
client PEP 'ClientJF' doesn't match any DynamicConfigPolicyLoad
statement, using defaults for all disciplines
```

See IP Diagnosis for more details.



You can help improve the quality of IBM Education Assistant content by providing feedback.

Trademarks, copyrights, and disclaimers

The following terms are trademarks or registered trademarks of International Business Machines Corporation in the United States, other countries, or both:

IBM z/OS

Product data has been reviewed for accuracy as of the date of initial publication. Product data is subject to change without notice. This document could include technical inaccuracies or typographical errors. IBM may make improvements or changes in the products or programs described herein at any time without notice. Any statements regarding IBM's future direction and intent are subject to change or withdrawal without notice, and represent goals and objectives only. References in this document to IBM products, programs, or services does not imply that IBM intends to make such products, programs or services available in all countries in which IBM operates or does business. Any reference to an IBM Program Product in this document is not intended to state or imply that only that program product may be used. Any functionally equivalent program, that does not infringe IBM's intellectual property rights, may be used instead.

Information is provided "AS IS" without warranty of any kind. THE INFORMATION PROVIDED IN THIS DOCUMENT IS DISTRIBUTED "AS IS" WITHOUT ANY WARRANTY, EITHER EXPRESS OR IMPLIED. IBM EXPRESSLY DISCLAIMS ANY WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NONINFRINGEMENT. IBM shall have no responsibility to update this information. IBM products are warranted, if at all, according to the terms and conditions of the agreements (for example, IBM Customer Agreement, Statement of Limited Warranty, International Program License Agreement, etc.) under which they are provided. Information concerning non-IBM products was obtained from the suppliers of those products, their published announcements or other publicly available sources. IBM has not tested those products in connection with this publication and cannot confirm the accuracy of performance, compatibility or any other claims related to non-IBM products.

IBM makes no representations or warranties, express or implied, regarding non-IBM products and services.

The provision of the information contained herein is not intended to, and does not, grant any right or license under any IBM patents or copyrights. Inquiries regarding patent or copyright licenses should be made, in writing, to:

IBM Director of Licensing
IBM Corporation
North Castle Drive
Armonk, NY 10504-1785
U.S.A.

Performance is based on measurements and projections using standard IBM benchmarks in a controlled environment. All customer examples described are presented as illustrations of how those customers have used IBM products and the results they may have achieved. The actual throughput or performance that any user will experience will vary depending upon considerations such as the amount of multiprogramming in the user's job stream, the I/O configuration, the storage configuration, and the workload processed. Therefore, no assurance can be given that an individual user will achieve throughput or performance improvements equivalent to the ratios stated here.