

Mediascope Programmers' Guide

Table of Contents

Problem Definition	2
Overview	2
Architecture of Mediascope	2
Mediascope Front End	2
Mediascope Intermediate Layer/Back End	2
Language/Platform Considerations	3
Software Architecture	3
Architecture of Mediascope Back End	3
Mediascope Back End Overview	3
“Translate” package	3
“Communicate” package	5
“Parse_XML” package	5
“Command_Center” package	5
Software Requirements	7
Overview of Software Requirements	7
Detailed Examination of Software Requirements	7
Easy configuration and use	7
Useful and Visually Attractive Display	7
Easy HTTP Router Interfacing	7
Modularity/Extensibility	7
Platform Support	8
Memory and Performance Impact	8
Mediatrace Demand Caused By App	8
Footprint of App on Host Machine	8
Packaging Considerations	9
Configuration and Restrictions	9
Testing Considerations	9

Problem Definition

Overview

The mediatrace technology from Cisco is a valuable tool to monitor and troubleshoot network traffic (http://www.cisco.com/en/US/docs/ios/media_monitoring/configuration/guide/mm_mediatrace.html). Mediascope, shown in Figure 1, provides an easily accessible platform which can be used to represent a visual, customizable view of network traffic via a user-friendly interface and a colorful, interactive set of graphs.

Architecture of Mediascope

Mediascope is a web-based application written in Adobe Flex Flash. It communicates with the initiating Mediatrace node via Web Services Management Application (WSMA). The architecture follows a simple three-tiered model—the front end (presentation layer), which lays out the GUI components, receives input from the user, and updates the visual display, the back end, which will perform two-way Mediatrace communication with the initiating node using a combination of Flash and PHP, and the middle layer, which is responsible for interfacing between the high-level requests of the front end and the low-level functions of the back end. Because the intermediate layer is responsible primarily for facilitating communication between the front end and the back end, in many cases in this document we will not refer to it directly, but rather include it in our discussions of the back end.

Mediascope Front End

The user interface of the app allows the user to perform the following actions:

- Log in to a Mediascope account to manage their saved Mediascope sessions, or create a new account if they do not have one.
- Establish connection to initiating node via WSMA by providing ip address of initiating node as well as username and password.
- Select a UDP/TCP flow going through a given initiator node to schedule a Mediatrace session for. Alternatively, enter the information for the flow manually.
- Configure flow detection on any interface on the initiating node.
- Schedule Mediatrace sessions with parameters gathered from a tree structure. Specify an action to associate with each parameter, which will define how it is displayed (these actions are currently coloring, sizing and plotting). For example, the user could specify that they would like to schedule a Mediatrace session with a CPU metric, and then ask to plot CPU usage on the y axis of the graph.
- Specify type of Mediatrace session to create (One-Time or Periodic)
- Save a Mediascope session to file, or load a previously saved Mediascope session.
- Generate a graph based on the previously specified parameters and actions.

Mediascope Intermediate Layer/Back End

The back end of Mediascope is responsible for establishing a connection with the specified initiating node, and translating the user-specified information into WSMA_EXEC and WSMA_CONFIG commands. At least one of these commands is a “show” command, which ultimately provides the data with which to render the visual network display. It then sends these commands to the initiating node via an HTTPService through a PHP proxy. Upon receiving the XML response to the “show” command, it parses the XML response for the user-specified information, and passes that information to the front end so that it may be used to update the display.

Language/Platform Considerations

Mediascope is an app built in Flex Flash. Flex is similar to traditional Flash in many respects—it runs client-side as an SWF in Adobe Flash Player in a web environment or in Adobe AIR in a desktop environment. In fact, anything that can be done in Flex can also be done in traditional Flash. The reason Flex was used over traditional Flash is that it contains a rich suite of XML-based visual components (such as trees, labels and data grids) that are easy to lay out and use. At the same time, it can leverage the power of Flash ActionScript, a full-featured object-oriented language with C-style syntax, similar to Java. Unlike Java, however, ActionScript was designed specifically with animation and vector graphics in mind, which makes it an ideal candidate for the kinds of tasks Mediascope needs to do—namely, present a rich user interface and provide strong data visualization.

The vast majority of Mediascope could be run either as an AIR or web app—really the only platform-dependent class is GrabFile, which performs the File I/O operations. Mediascope is currently launched as a web app, to allow the maximum degree of exposure and ease of use. It contains an account control system, and requires that users create an account (only a username and password are required) before use, so that session information may be saved to their profile without disrupting others.

In addition to Flex, Mediascope does include a small number of PHP scripts to perform WSMA communication and file I/O. This is necessary not because of technical limitations within Flash, but because of security sandboxing restrictions. Because the PHP scripts reside on the same server as the Flash app, they may be accessed directly, and then they in turn may perform any I/O necessary without having to worry about security restrictions.

Software Architecture

Architecture of Mediascope Back End

Mediascope Back End Overview

The back end of Mediascope is responsible for translating the raw input it receives from the front end into WSMA_EXEC and WSMA_CONFIG commands, and sending those commands to the initiating node via an HTTPService through a PHP proxy.

The back end configures Mediatrace sessions and, once those sessions are configured, executes “show” commands to retrieve the Mediatrace results. The back end then takes this output, imports it into an XML object, then parses that XML object for the data it requires. It then provides this formatted data to the GUI for display in the chart.

The back end is also responsible for I/O with files on the server (for saving and loading session templates, for example). This also happens via a PHP proxy.

The remainder of this section will break these tasks into smaller groups (where each group is a package) and explain their function in-depth. See Figure 1 for more information.

“Translate” package

The “translate” package is comprised of only the “Translate” class. This ActionScript class is responsible for taking unformatted user input provided by the UI and converting it into IOS CLI commands (Mediatrace commands and otherwise). It does not send these commands to the CLI; it only creates the commands themselves. In other words, rather than creating the text for a command and executing it directly on a router, the “translate” package only creates the text for the command, and leaves it up to another package (the “communicate” package) to execute the command over WSMA. These commands are encased in WSMA request wrappers. Each line of both exec and config commands begin with “<cmd>” and end with “</cmd>”.

Every function in “translate” is static, and most take as a single parameter a hash with all the parameter information from the GUI and returns a string of commands used to configure or execute a certain task. To learn more about the “translate” package’s functions, see the package’s ASdoc.

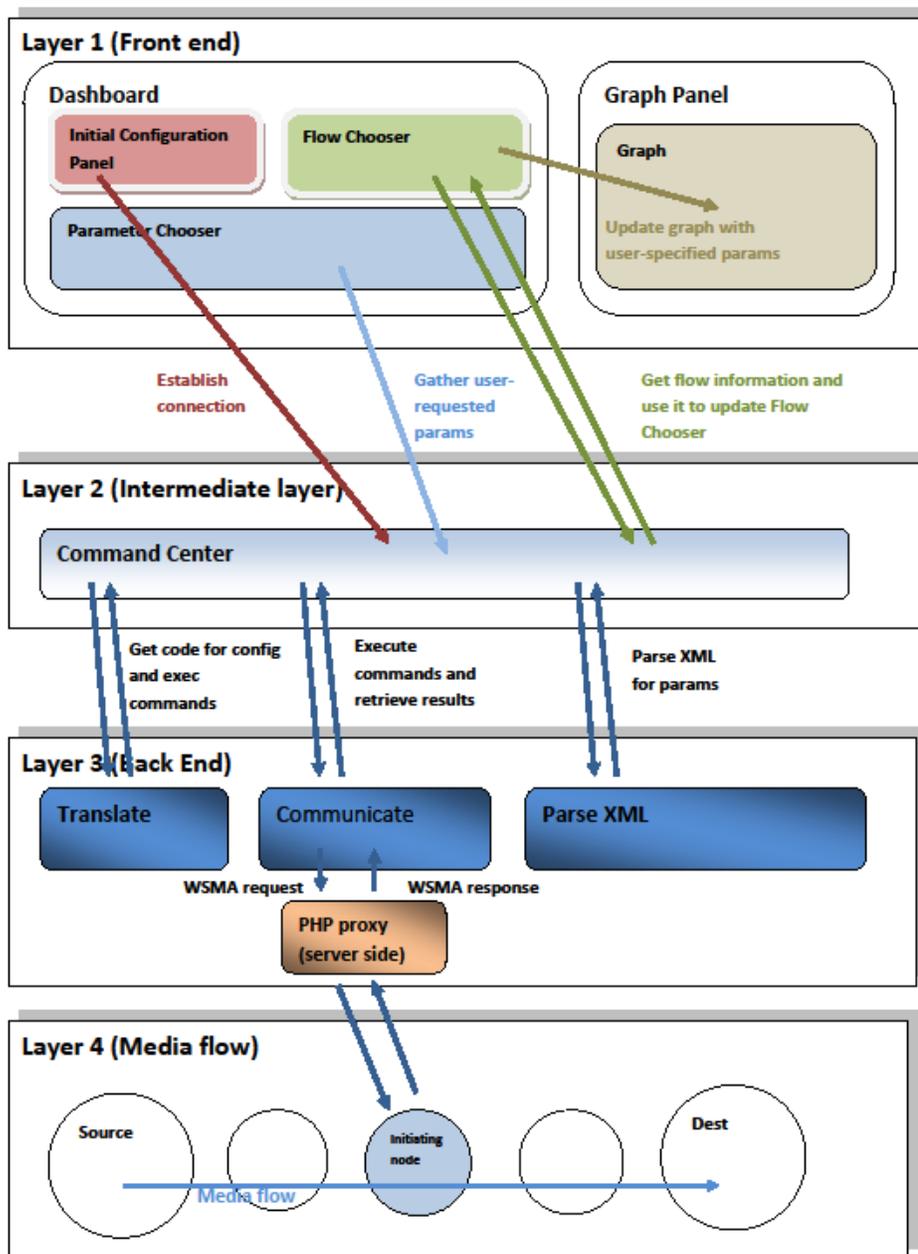


Figure 1. Mediascope Architecture

“Communicate” package

The “communicate” package is responsible for sending WSMA requests to and receiving WSMA responses from the initiating node, as well as performing file I/O with the host server for the app. These tasks are performed by the “WSMATalk” and “GrabFile” classes, respectively.

The WSMATalk class communicates with the initiating node as follows: It receives as input the syntax for a given “config” or “exec” command from the “translate” package, then wraps this string with a WSMA header and footer. It then creates an HTTPService object and attaches to that object’s “request” object a number of data members. One of these is the aforementioned concatenated WSMA request string. Others are the IP address, username and password of the initiating node. The module POSTs this object to a PHP proxy living on the host server, which encrypts the credentials, then execs a cURL request to the IP address of the router. The router executes the request, then returns its response back to the proxy, which echos it back to WSMATalk. (If there is a cURL error—i.e., the router does not respond—this will also be echoed.) When WSMATalk receives this response, it calls another function that strips out the WSMA header and xmlns, then attempts to parse the response to an XML object. If this is successful, it calls a function in CommandCenter (which CommandCenter had previously set) with the object; if not, it calls the error function.

Because the response from the router is asynchronous, any call to this class must include not only the command and command type (“exec” or “config”), but also a pointer to a function to call with the XML result of the request, and a function to call in the event of an error. (In some cases, the result of the command may not matter. In that case, either one or both of the function pointers may be set to null.)

The other file in this package, GrabFile, performs simple reads and writes to the host server filesystem. As with WSMATalk, none of this is done directly through Flash; rather, Flash POSTs the needed information to PHP scripts on the server, which then perform the file I/O.

“Parse_XML” package

The “parse_xml” package is responsible for parsing an XML response from Mediatrace for a given parameter within a given session, and returning the value associated with that parameter.

The “parse_xml” package has only one file—the ParseXMLToArray class. This class contains a pointer to an XML object which may be set by another object, and then defines functions that will perform specialized searches within this object. Because CommandCenter currently returns a list of all Mediatrace sessions when it retrieves session stats, it is necessary to have the capability to parse for a specific value within a specific session, a functionality this class allows. The class also supports the ability to parse all the information from a single node across all the sessions that are generating a given graph, an important functionality for generating node information when a node is hovered over in a graph.

“Command_Center” package

The “command_center” package is responsible for interfacing between the high-level requests of the front end “layout_ui” and “graph_components” packages and the low-level functions of the “translate”, “communicate” and “parse_xml” packages. Rather than requiring the front end packages to explicitly call “translate” and “communicate” every time they have to make a request, the “command_center” package contains high-level functions which the front end can call which will handle the translating and communicating tasks.

This package is comprised of three classes: “CommandCenter”, “Constants” and “ScreenscapeFlows”.

The CommandCenter class is responsible for performing the following functions:

1. Open/close the connection with the initiating node. (This function is called by the “InitialConfigPanel” module in the front end). The IP address of the node, as well as the username and password to the router in question, are required. The function simply calls executes a “show version” on the router (through the “translate” and “communicate” packages) and responds back to the “InitialConfigPanel” with the result. The result is either the text of “show

version" if the request was successful, or an error code if it was not. CommandCenter parses this response to an int (where a String becomes 0) and sends that to the "InitialConfigPanel", which updates its status message based on it. Regardless of whether the connection attempt was successful, the credentials in WSMATalk which will be used for future accesses are set to the value of the attempt.

2. Update the flow chooser. This function is called by the "open/close" function whenever the user enters a new initiating node, or it may be called by the user to refresh the flow list. It calls the "communicate" and "translate" packages to gather this information via passive monitoring. It then parses the flows into a 2-D array, which it sends to the "Flow chooser" panel.
3. Configure Mediatrace sessions for all parameters chosen in the front end. (This function is called by the "graph_components" package.) It takes a two-dimensional array of requested parameters, in which each row contains a hash with the following information gathered from the GUI: path name, flow name, source IP, source port, destination IP, destination port, IP protocol, profile name, profile type, metric-list, session name, session life, session start time, session-params name, session-params response timeout, session frequency, session-params inactivity timeout. The function will search for duplicate profiles, and if it finds any, it will delete that profile's hash from the array. It will then call "translate" and "communicate" to create Mediatrace sessions based on this information, and schedule these sessions. (When configuring these sessions, the configuration commands are sent in small groups, one group at a time, rather than being concatenated and then sent in one request. Although this latter approach would be simpler, it resulted in a high degree of WSMA errors during testing. As a result, CommandCenter instead keeps an array of function pointers to functions in "Translate", where all the functions together configure a complete Mediatrace session. It loops through this function array, calling each function in turn only once the previous command has completed. This approach seems to eliminate WSMA errors.)
4. Retrieve the parameters from the previously scheduled Mediatrace sessions, when these parameters are ready. The first call to retrieve is made after whatever duration of time the Mediatrace timeout was set to, plus a small "delta" (a short period of time—currently set at 1 second—to allow Mediatrace data to be retrieved after the timeout). After this first "show Mediatrace session stats" command, this command is repeated until all the results are available. It then calls the "finishSubmitParams" function in NetworkLayout to generate a graph based on this information. (Note: with "show Mediatrace session stats", CommandCenter retrieves information from all currently running Mediatrace sessions, not just the ones it has just scheduled. This is not as efficient an approach as it could be, especially if there are many other Mediatrace sessions running on the initiating node, but it is the simplest and suffices for the purposes of the app. In the future, it would be good to take a more targeted approach to Mediatrace data retrieval—that is, ask for one specific Mediatrace session at a time. This would not be too difficult to do: just ask for the first Mediatrace session instead of asking for all of them, and when that is available, create an XML object and add those Mediatrace session results to it. Then ask for the next session, and add it to the XML object when its data is available, and the next, and so on. Once the XML object has as many children as sessions, return it.)

The "Constants" class is responsible for keeping track of various static constants used throughout the application, as well as certain universal lookup functions (such as looking up a session number based on individual offset and metric list).

The "ScreenscapeFlows" class is responsible for taking raw flow CLI output from "show perf traffic history ip any any" passive monitoring output and converting into any array of hashes that can be ready by FlowChooser.

Software Requirements

Overview of Software Requirements

Mediascope must be very strong in certain areas—it must be easy to configure and use, must provide a clear, useful and visually attractive display, must be able to interface with a router quickly and easily, must be flexible for future modifications to Mediatrace and WSMA, and should preferably run on as many platforms as possible. The remainder of this section discusses these requirements in-depth.

Detailed Examination of Software Requirements

Easy configuration and use

Mediascope should be as easy to configure and use as possible. Whereas a Mediatrace session can take many lines of configuration code to set up, and still more to execute as desired, this app should require as little user input as possible to produce the desired result. In order to do this, the app requires a minimum of user effort to set up—all that is needed is the address of the initiating node, a choice of path, the type of session, and the parameters to watch and how to visualize them. No knowledge of IOS Mediatrace syntax is needed; all the information can be provided in text boxes and dropdowns.

Once the information needed is gathered, tailoring the display to fit the user's needs is easy too. Even once the graph has been displayed, the user can edit the display to emphasize different parameters, without going through the lengthy process of polling again for data.

Useful and Visually Attractive Display

As the entire thrust of this project is to produce a topological view of the network that is in some way more useful than straight numbers, it is very important that this app be able to deliver graphs that each convey as much information as possible in the space allotted, and that this information be easily accessible. The graph generated for each Mediascope session is specifically tailored to display as much information as possible in a single space—by plotting hops on the X-axis, it can display one Mediatrace parameter as node Y-position, another as node size, and an unlimited number of parameters as color. Flex Flash itself is a language optimized for this sort of display—it is marketed by Adobe specifically as being strong in graphics in general and data visualization in particular. For more information on the graph portion of the app, please see Section 7 – “End User Interface/User Experience”.

The display itself is designed to highlight the following characteristics of the network:

1. The path itself [in real time and over time]
2. A particular parameter over the path
3. Multiple parameters over the path [both static and dynamic]
4. One parameter over time

Easy HTTP Router Interfacing

The app must be able to communicate with an external router using WSMA. Flash contains built-in support for HTTPRequests, which are all that are needed to communicate with WSMA, but security sandboxing restrictions prevent it from communicating with any external device that does not explicitly list it as a “friend”. Luckily, PHP's cURL module has no such restrictions, and Flash/PHP communication is easy—instead of POSTing a WSMA request directly to the router, POST it to a PHP script instead along with the rest of the router information and credentials, and PHP will repost this to the router via cURL and give the response back to Flash asynchronously when it arrives.

Modularity/Extensibility

Mediatrace is still a work in progress, and will likely see many changes in the coming months and years. As Mediatrace evolves, this app must be able to evolve with it. To ensure the least painful updating processes possible, the class that actually produces Mediatrace commands—the “translate” package—is

a separate entity from the user interface and communication architectures, making it easy to change in the future without the need for extensive revisions in the rest of the code.

Furthermore, it should be possible in the future to improve the way users choose parameters to view. Currently, this requires explicitly creating sessions, profiles, metric lists and parameters to watch, but in the future, only the parameters should be required—figuring out what metric lists and profiles those parameters are associated with, and scheduling the appropriate Mediatrace sessions, should all be done within the “translate” package. Luckily, again due to the self-contained nature of this package, simply adding this functionality within “translate” should not be difficult.

In the future, it is likely more Mediatrace metric-lists will be added, or more parameters will be added to existing metric-lists. Mediascope populates the ParamChooser tree directly from an XML file (ProfileData.xml) living on the server, which contains all the profiles and metric-lists Mediatrace currently offers, as well as the parameters associated with each. If the only change is that more parameters are being added to existing metric-lists, the only thing that needs to be changed is that the new parameters need to be added to ProfileData.xml. If a new profile or metric-list is added, ProfileData.xml again needs to be changed, and the Constants class in CommandCenter needs to be updated with the new metric-list so that the session number can be retrieved from it.

Platform Support

We have tried to make Mediascope as platform independent as possible. By making this app a Flash-based web app, we are ensuring it will run in every browser on nearly any operating system (Linux/Windows/Mac/Android). Because Flash is not interpreted by the browser, it is browser-independent, and thus there is no need to worry about browser compatibility. And since Flash has about 90 percent market penetration, there will be very few users who cannot access the app.

Memory and Performance Impact

Mediatrace Demand Caused By App

Mediatrace does cause additional overhead in the routers it polls, so it is important that the Mediatrace activity invoked by this app remain at the minimum level necessary in order to insure proper function of the app. Mediascope also reduces the size of the Mediatrace footprint by scheduling no more Mediatrace sessions than necessary (for example, if the user asks for five parameters that are all within one profile and metric-list, Mediascope will schedule just one session, not five). Furthermore, because the user can choose the metrics they would like to follow, if they wish to reduce the app’s impact on the network, they can simply choose to follow fewer parameters or parameters that are less costly to obtain. Besides these parameter-based sessions, the only other interaction the app will have with the router is to gather flows, which happens only when users request it.

Footprint of App on Host Machine

Because Flash runs client-side, there is definitely a memory impact on the host machine while the app is running. Because performing graph updating for a periodic session stores much information from old data-gathering and some visual components from past graph views, there is a memory impact in RAM over time, especially if the graph is being updated frequently. Besides storing some costly elements—arrays, XML objects, UI components—the memory impact is exacerbated by a relatively lethargic Flash Player garbage collector.

This is definitely an issue that should be addressed in the future, and there are several ways of going about doing it. The first is to aid the garbage collector by explicitly removing every visual element not in use from the display tree (Mediascope currently tries to do this whenever possible, but it should be double-checked). Another improvement would be to limit the number of past graphs saved in periodic

session mode, or better yet, save old graph information to disk, possibly compressing to save additional space. Still another is to limit the number of graphs that can run in periodic mode in parallel concurrently.

Packaging Considerations

Mediascope is launched on a web server, with account management built in so that users may save their sessions a specific profile, to avoid interfering with other users' saved sessions.

Configuration and Restrictions

To be able to build Mediascope, you will need to download Adobe Flex SDK 4.0 or higher and Adobe Flash Player 10 or higher (you probably want the debugger extension as well). Both are free. For an IDE, you can try either Flex Builder 4 (3 might work as well), which is sold by Adobe, or FlashDevelop, which is free. Move copy the Mediascope source folder into your source folder and build. (Make sure the compiler is set to 4.0 or later in your IDE; otherwise it won't build properly.) It's OK if you see warnings when building, but there should be no errors.

That's all you need to do from the Flex side. Since Mediascope also uses PHP, you'll need to have PHP running on the same machine you're running your app on. If you're using a PC, I recommend installing Apache and running PHP off of that. You can use localhost port 80 or 8080; either way, check to make sure the url to the proxies are correct in WSMATalk and GrabFile.

Note that PHP needs to be able to access the cURL module. You can configure it yourself, or copy my configuration on.

Once you have PHP and apache set up, move the contents of my "apache" folder into whatever folder you're using as root. Then fire up Mediascope and you should be done.

Testing Considerations

The back end of the app was tested using Pagent traffic generator; the front end was tested primarily using synthesized XML data