Realtime publishers

The Definitive Guide™ To

Application Performance Management

sponsored by

Greg Shields

Copyright Statement

Chapter 3: Understanding APM Monitoring

Its Memorial Day weekend, and the volleyball game is about to start. Coals in the barbeque grills are coming up to temperature, and the weather is cooperating to make it a perfect late-spring day. Prepping for the BBQ, friends and family are milling around, swapping stories, and relaxing in the sun.

Then it happens, always just as the rest of life is smooth-sailing and work is the last thing on the mind: BEEP, BEEP.

"Uh-oh, there goes your vacation-ending device," says a friend to John Brown, IT manager for TicketsRus.com. John looks down to read the text now displayed on his pager, "I can see it in your face. Something's down, you probably don't know what it is, and the only way to figure it out is to set down that cold one and march right into work."

John shakes his head at the pager, "This thing is killing me. Since we installed the new monitoring system, I swear I'm getting pages like this every couple of days. Half the time it's nothing. The other half the time it's something completely different than what shows up on this stupid thing. You know, monitoring is great, but this kind of monitoring is taking my life away from me."

"You going in?"

"Yep. Got to. If this is correct, the problem could be a big one, and you know what happens when fans can't get their tickets..."

John's friend jokes, "We don't want that to happen! I still remember that day when I and everyone else couldn't get tickets to the big game through your site. That problem was so bad, it made the news!"

"Don't remind me," grumbles John, remembering that painful event in his past. A bug in the code between the inventory and e-commerce subsystems for TicketsRus.com decided to rear its ugly head just as tickets were released for the Finals. The bug, which for some reason only caused problems at high loads and for certain types of events, had been introduced earlier in the year with a software update. Because user loads had been light for the following months, it took literally days to track down the error. TicketsRus.com, this team's sole source for Finals tickets, was criticized by its suppliers and even the press. It nearly lost a major source of income out of the problem.

To rectify the problem, John's boss Dan mandated that a monitoring system be put into place. Since then, John has come to regret his selection of monitoring system, an inexpensive but limited solution that delivered alerts on server outages and not much else. The net result is that John's nights got a lot more sleepless and an e-commerce system which "felt" fine before now alerted him and his teams on a near-constant basis.

John tells his friend as he heads for his car, "I've gotta' run. Take care of the burgers for me. Who knows when I'll be back…"

John's problem is not uncommon. There's a fundamental problem intrinsic to most traditional monitoring solutions. Namely, these types of solutions are almost completely reactive in nature. Using a traditional network monitoring solution, the system alerts on the problem only after the problem occurs. Such a system looks constantly at your network to identify where a change has occurred that signals a problem—a device stops responding to "ping" requests, a network connection slows down, a server's processors become overused by a particular process. When that change occurs, it sends an alert to administrators, notifying them of the problem.

This information is excellent for knowing when something isn't right with your IT infrastructure. It gives you the information you need to know that a problem exists. But this kind of information is solely limited to answering the question, "What happened?" Knowing that a particular server, service, or device appears down is one thing. Understanding exactly why it went down is quite another.

That's not to say that reactive monitoring isn't useful in an IT environment. In fact, nothing could be further from the truth. Consider the story that started out this chapter: Without some form of monitoring in place, John would never have known that something was amiss in his data center. Prior to the Memorial Day incident, not having that monitoring in place would have easily turned a small problem into a big one. A simple outage could have gone unnoticed for minutes or hours while TicketsRus.com's customers were unable to purchase the products they needed at the time they needed it.

Traditional network monitoring is an excellent solution for organizations that operate in the early phases of IT maturity. This technology gives them a basic understanding of the 1s and 0s passing back and forth across their network and within its connected systems. Yet traditional network monitoring can only go so far. As businesses and their IT organizations mature in capabilities, their philosophy of service delivery must mature as well. Although up/down monitoring might work well for a simple environment, its level of granular detail is wholly inadequate to keep an online, 24 × 7, highly-available storefront open for customers.

In effect, *added success begets added due diligence*. Like in the case of the Finals mentioned earlier, as your suppliers and customers rely on you for greater things, they expect greater things as well. Application Performance Management (APM) and its advanced monitoring integrations enable you to provide that greater level of service.

The Evolution of Systems Monitoring

The previous chapter of this book spent a lot of time discussing the concepts of IT organizational maturity. Although that conversation has little to do with monitoring integrations and their technological bits and bytes, it serves to illuminate how IT organizations themselves must grow as the systems they manage grow in complexity. As an example, a Chaotic or Reactive IT organization will simply not be successful when tasked to manage a highly-critical, customer-focused application. The processes, the mindset, and the technology simply aren't in place to ensure good things happen.

To that end, IT has seen a similar evolution in the approaches used for monitoring its infrastructure. IT's early efforts towards understanding its systems' "under the covers" behaviors have evolved in many ways similar to Gartner's depiction of organizational maturity. Early attempts were exceptionally coarse in the data they provided, with each new approach involving richer integrations at deeper levels within the system.

IT organizations that manage complex and customer-facing systems are under a greater level of due diligence than those who manage a simple infrastructure. As such, the tools used to watch those systems must also have a higher level of due diligence. As monitoring technologies have evolved over time, new approaches have been developed that extend the reach of monitoring, enhance data resolution, and enable rich visualizations to assist administrative and troubleshooting teams. This chapter discusses how this evolution has occurred and where monitoring is today. As you'll find, APM aggregates the lessons learned from each previous generation to create a unified system that leverages every approach simultaneously.

Early Network Management

In the beginning, there were only a few computers. Those computers each accomplished all the tasks necessary for its stated mission. Then came the "network," which brought about an explosion of interconnections among computers. These computers worked together and communicated with each other to distribute their processing load. The network brought about a systemic shift in mindset from centralized to distributed processing. It also dramatically increased the number of moving parts required for an application or service to function. With data processing now occurring across more than one piece of hardware, the health of each individual component directly impacts the success of the entire system.

Simple Availability with ICMP

The earliest network management solutions were singularly focused on the availability of computer systems and the network that connected them. As Figure 3.1 shows, the earliest attempts at measuring the availability of a component were coarsely focused on whether that server responded to an ICMP "ping" packet.

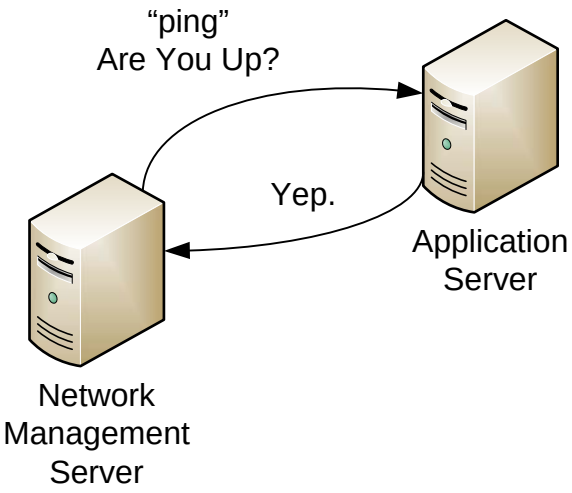

Figure 3.1: Early network monitoring was singularly concerned with system and device availability.

This solution works well for low-criticality environments because it is elegantly simple. If I ping the server every 2 minutes, I'll know that that server has gone down no greater than 2 minutes after the outage occurs. Implementing basic availability monitoring is a key step for organizations that want to move from Gartner's Chaotic phase ("my users let me know when the server is down") to its Reactive phase ("my 'ping' script lets me know when the server is down").

But basic availability metrics can only go so far. Servers that are experiencing a processor spike condition may be wholly incapable of processing useful data. Responding to an ICMP "ping" request is an extremely low-level interrupt that requires virtually zero processing power. Thus, even an unhealthy server can usually successfully respond to a ping request. As such, basic availability metrics generally cannot identify when a server is not down but merely hung.

Richer Information with SNMP

Focused initially on networks, more information was later deemed necessary to identify an environment's health. Due to the complexities of network traffic management, networks were one of the first parts of the IT environment to gain more granular information. One early solution began with the development of the Simple Network Management Protocol (SNMP) back in the late 1980s.

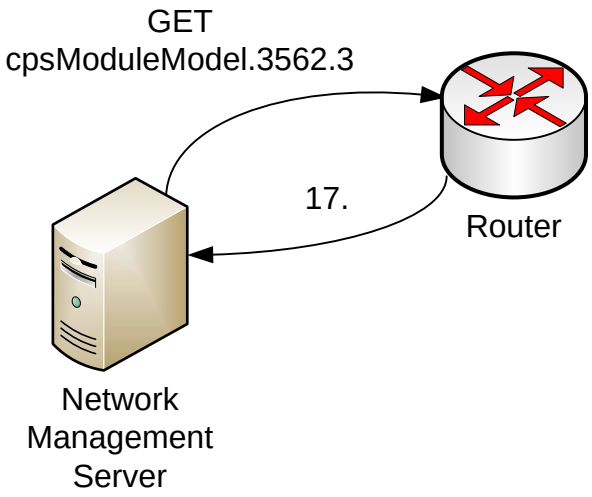

Figure 3.2: With SNMP, any SNMP-aware information can be gathered through a request/response interaction.

With SNMP, a framework was established for requesting and receiving detailed information from networked devices. That framework operates on a request/response basis, with a network manager requesting information from an onboard SNMP agent through a network call (see Figure 3.2). The network manager identifies the category of information requested by its Management Information Base (MIB) Object Identifier (OID). This OID is a unique identifier for the specific piece of information being stored by the client. For example, in Figure 3.2, the network management server is attempting to GET the information located at OID *cpsModuleModel.3562.3*. Globally unique across all devices, the contents of that OID can be anything:

- Network statistics
- Device configuration information
- Sensor information
- System or device performance metrics

As configured by an administrator, it is the job of the network manager to determine which information is interesting and should be polled. That information is stored within the network management system's database for later review by an administrator. The network management system is configured to alert administrators when information is received on inappropriate behaviors. Similarly, clients can alert the network management system unilaterally through an SNMP trap when special conditions occur that require more immediate attention.

Device Details with the Agent-Based Approach

SNMP was an excellent solution for determining information about devices all across a network, and is still heavily used today. However, SNMP's design wasn't without its own architectural limitations. Although SNMP remains in use today for the monitoring of network hardware, that original network focus limited its acceptance in the realms of servers, server OSs, and applications.

There are numerous reasons for SNMP's limited scope and reach here. In order for an SNMP-enabled network manager to gather information from any element on the network, that element must have SNMP awareness. Thus, every device, operating system (OS), application, and service must internally convert its own instrumentation data into the format that SNMP understands. Also problematic is the poll-based nature of SNMP. SNMP is configured to poll devices for their information on a regular basis, with the network management server the source of those polls. This can create a bottleneck as the level of monitored devices scales and limits the resolution of its data. Finally, until only recently, SNMP lacked key security features.

To combat SNMP's intrinsic limitations within non-network devices, OS and application manufacturers designed agent-based solutions. These solutions gathered otherwise unavailable availability and performance data from servers. In order to successfully access this data on systems, these agent-based solutions required the installation of client software. Here, the client's job was to leverage its on-system privileges to gather necessary data and eventually transfer it to a centralized monitoring solution (see Figure 3.3).

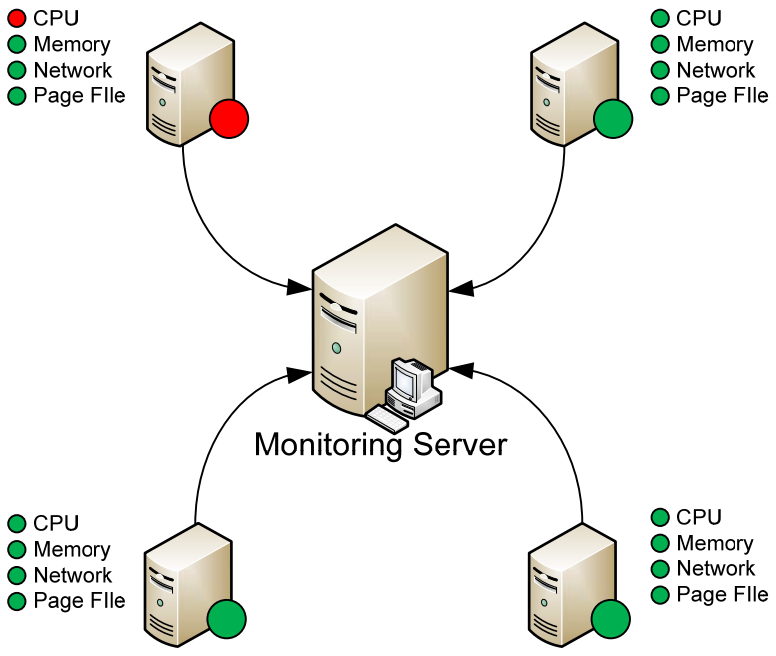

Figure 3.3: Agent-based solutions leverage on-board clients to gather and transfer monitoring information to centralized monitoring servers.

Agent-based solutions are predominantly device-specific, focusing on server and application metrics for elements that are available on the system. Their inline installation means they have a deep level of access to on-system event and performance metrics. Agent-based solutions can provide details about the activities on a system as well as their resource use.

As with SNMP solutions, agent-based solutions are in widespread use today. For servers, services, and applications, these solutions enjoy benefits over and above SNMP-based solutions because their information does not need translation into a format that is understood by SNMP. For example, if an Oracle database decides to store performance information one way, while a Siebel installation elects another method, both can be easily encoded into the agent software. The agent can collect this information irrespective of original vendor, source, or format, and translate it into a format that is useable by the central monitoring solution.

Agent-based solutions also enable a much greater resolution of data, enabling monitoring to scale with the needs of high-performance and high-criticality systems. The result is a high degree of data resolution associated with metrics on the individual system itself. Figure 3.4 shows an example of a graph that can be created out of such data, showing how the health of the server and an installed database are graphed over time. As later chapters will discuss, visualizations like this roll up low-level metrics to provide a high-level understanding of a component's health and service quality.

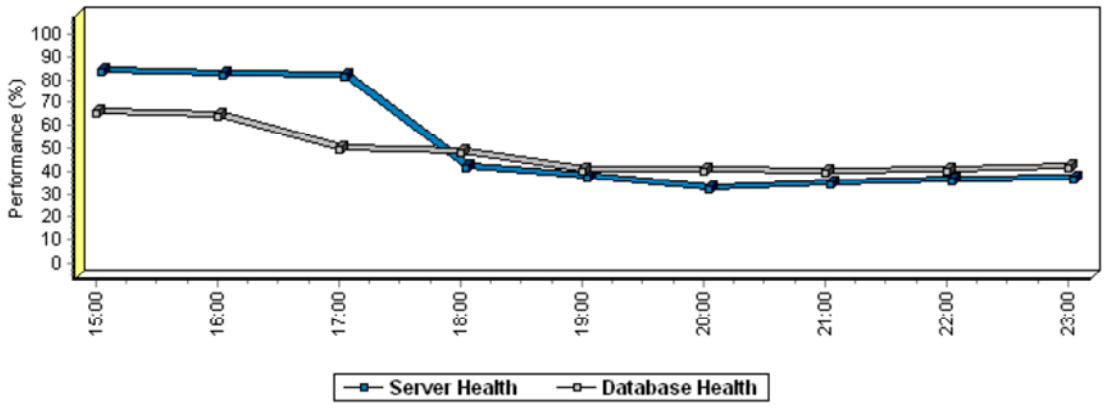

Figure 3.4: Multiple agent-gathered metrics can be consolidated to create an overall picture of a component's health.

Situational Awareness with the Agentless Approach

Agent-based monitoring solutions filled a critical gap in early methods. They provided a system-centric approach to gathering performance information. This system-centric approach enabled administrators to be alerted when systems experienced problems other than core availability, such as when performance behaviors degraded beyond acceptable thresholds. The agent-based approach also provided a more system-friendly framework for measuring performance across homegrown applications, with some solutions tying into code frameworks for additional exposure.

However, the strengths of the agent-based approach also give way to its primary weakness. A naturally system-centric solution, the agent-based approach only looks at information on that system itself and from the perspective of the system. Relating instrumentation data across multiple systems wasn't natively possible with an agent-based approach unless that relation was done at the central monitoring solution. This proved problematic. For many environments, the limitations of the agent-based approach grew more obvious as the count of hardware instances required to build an application or service grew in number. Most specifically, using only an agent-based, server-centric approach, it was impossible to visualize impacts *coming from the rest of the environment.*

The Impact of Externalities

Environment-based impacts aren't a new problem. If you've ever attempted to use the free broadband Internet at your local coffee shop on a busy day, you're familiar with the impact of external forces. Although your computer might be running perfectly, its performance on the network is greatly impacted by the Internet surfing of everyone else on that network. In this case, if your monitoring is limited to the agent on your computer, that agent has no capability to understand the problem because its scope is limited to just your system.

Relating this situation to a business application scenario, let's take another look at the simplistic system discussed back in Chapter 1. In that system, shown again in Figure 3.5, a number of elements integrate to provide a service for end users. However, in this second scenario, a network-based tape backup device is also on the network. Due to a misconfiguration by an administrator, a large backup has been initiated against the mainframe during the workday.

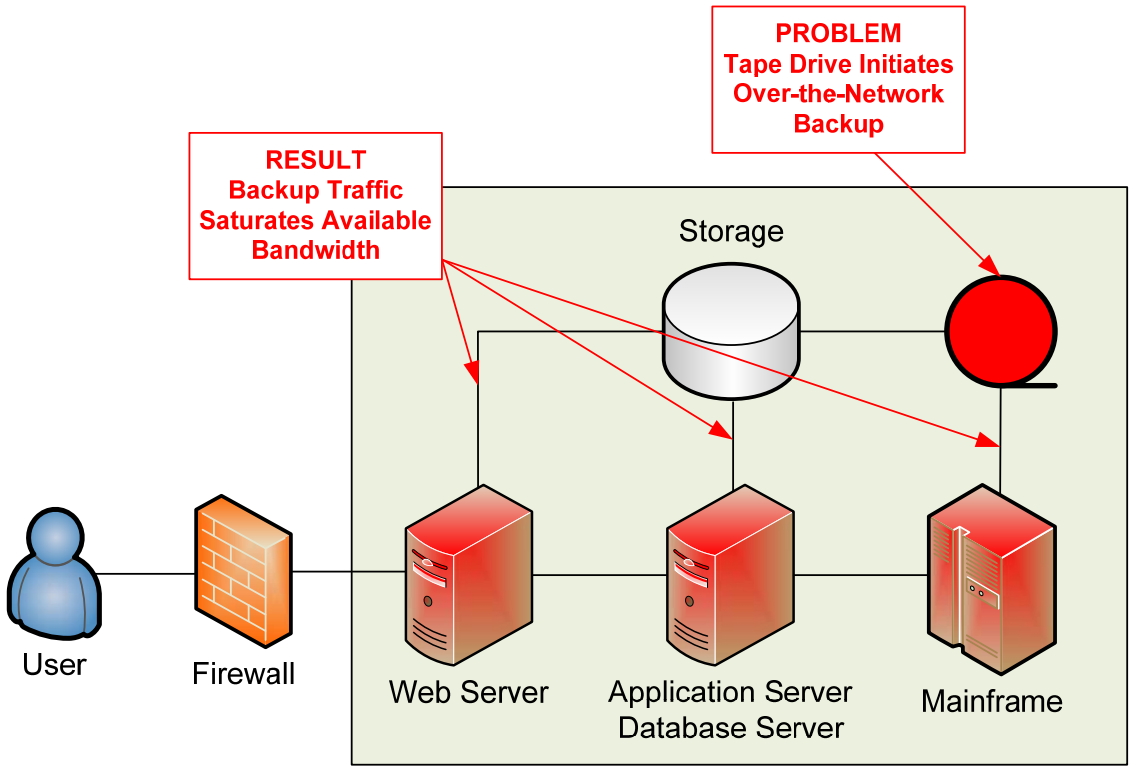

Figure 3.5: The actions of an unrelated device can cause performance problems on the monitored system.

Large-scale tape backups are usually scheduled to occur during times of low processing requirements. One reason for this is because the backup process can require an incredible amount of network bandwidth if not properly tuned or segregated to alternative networks. In this case, the entire customer-facing system is affected by a mistake made on a completely unrelated device. Information gathered by agents on each of the devices cannot easily show that a problem is occurring. Yet, the net result is a substantial reduction in performance across the entire system.

It is for situations like this that an agentless approach is additionally necessary. Figure 3.6 shows an example report from an agentless monitoring solution. This report shows that the primary consumer of network bandwidth is related to VPN traffic. The results from this report can and should be cross-referenced with information from agent-based visualizations to get a better situational awareness of network conditions.

-	Application	Type	KBytes	Packets
	Shiva VPN	UDP Port:2233	1547.6	3130
	HTTPS	TCP Port:443	745.7	1808
	HTTP	TCP Port:8080	277.4	784
	MS Print Spooler	OFR:/spoolss*	53.2	226
	UDP*	Internal:UDP	22.5	66
	Domain Name Server	UDP Port:53	10.3	86
	HTTP	Internal:TCP	9.0	20
	Low Volume Traffic	Internal:Data link	6.6	48
	UDP	Internal:UDP	6.5	20
	MS Browser*	Internal:UDP	3.1	13

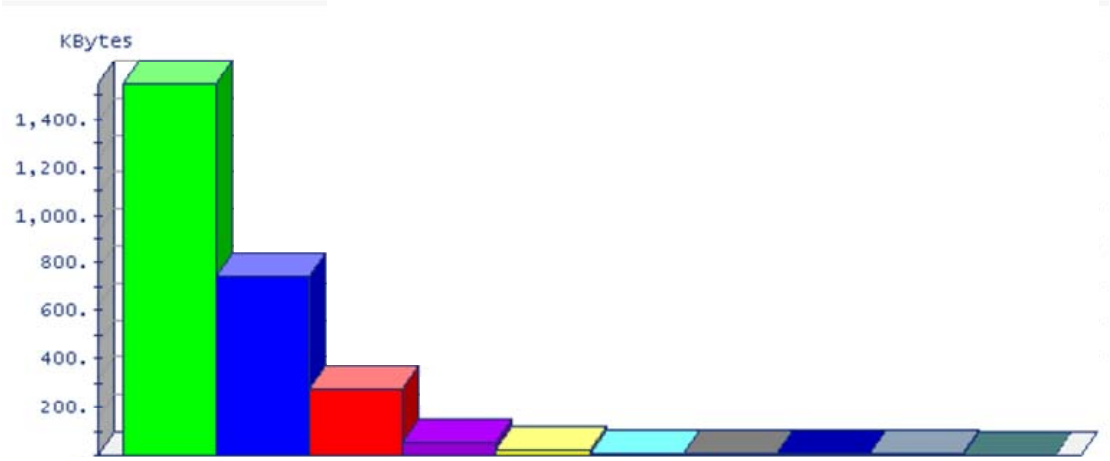

Figure 3.6: An agentless monitoring approach reports on aggregate traffic across the environment.

Although situations like this tape backup mistake are unlikely to happen in a well-managed production network, other environment behaviors can and do have impact on application performance. Perhaps an e-commerce system experiences a flood of requests, limiting the data it can successfully process in a particular period of time. Or, the overuse of a separate and unrelated system on a shared network impacts the performance of a customer-facing application. Only through the simultaneous use of both agent and agentless monitoring can an IT organization get a complete understanding of its environment's behaviors.

That's a Lot of Traffic!

As you can imagine, the level of traffic that such a system must monitor is enormous. With dozens or hundreds of devices communicating at extremely high speeds, the primary responsibility of an agentless solution is usually to determine what traffic can safely be ignored.

One of the ways that network monitors like the ones explained in this section limit that traffic is through the use of rules. These rules allow the device to safely watch for traffic based on characteristics. An effective agentless solution should include a large number of filters that can be used to create such rules. Those filters can relate to but are not limited to:

- IP address (unicast or multicast), or IPX socket
- TCP/UDP characteristics
- MAC address
- Microsoft RPC/DCOM ID
- Novell Netware SAP message
- Database ID
- SNA Application ID
- SOAPAction string within an HTTP request
- Sun RPC Program Number
- URL path
- Identified protocol

As is obvious, more options available for filtering traffic will mean greater performance in monitoring that traffic as it goes by. One primary step in implementing this kind of monitoring will be the characterization and isolation of the types of traffic you want to monitor.

Direct and Indirect Monitoring

The agentless approach leverages both direct and indirect network integrations in order to see this data as it crosses the network. Direct network integrations gather their information through the use of network probes. These probes are attached between devices at various parts of the network. They passively watch traffic as it goes by, reporting their findings back to the centralized network monitoring solution.

Indirect network integrations operate in a much different way. Rather than installing devices directly on the network, physically bridging connections between devices, indirect network integrations interface with specially-enabled network devices to directly gather their statistics. Common protocols such as NetFlow, JFlow, SFlow, and ipFIX function across different network components to gather flow-based network traffic information from devices.

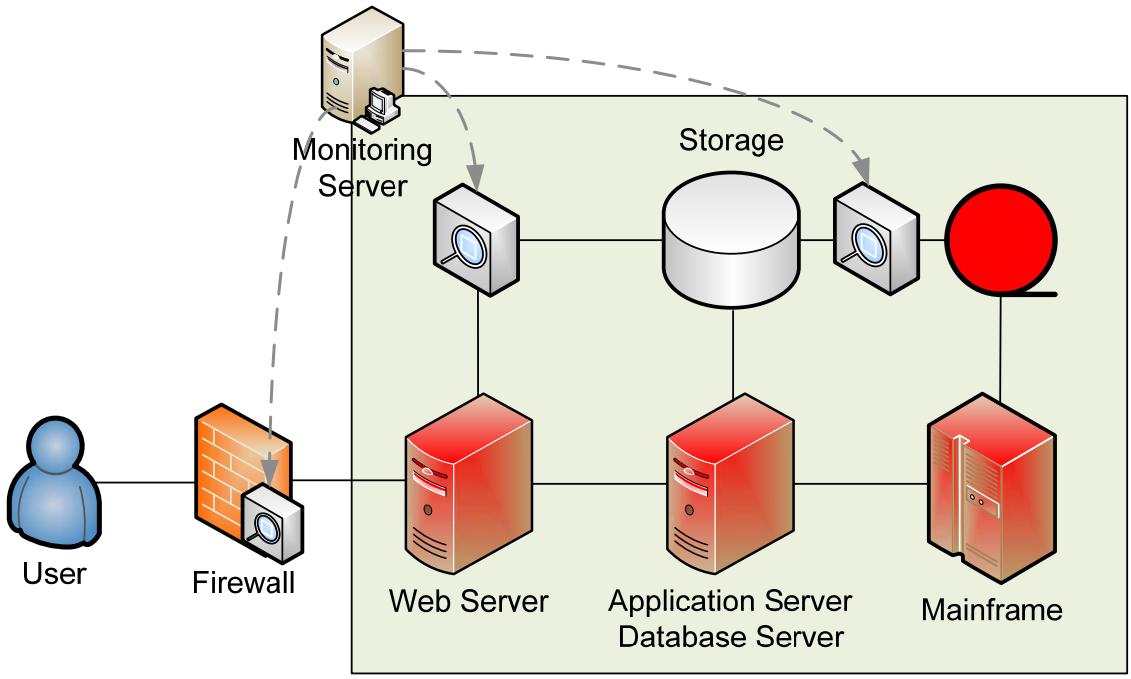

Figure 3.7: Network probes and on-device monitoring protocols such as NetFlow enable agentless monitoring across the entire network.

A singular difference between the direct and indirect methods is whether an actual device—the "probe" itself—must be physically installed between connections. Obviously, the installation and maintenance associated with physical probe devices adds an administrative burden to their use. However, probes can be installed virtually anywhere, making them highly flexible in heterogeneous networks. This is in contrast to the easy administration associated with indirect integrations. Devices must natively support such integrations, so not all areas of the network may be accessible. Figure 3.7 shows an example of both types.

Transaction-Based Monitoring

And yet even with these two types of monitoring integrations in place, mature environments still found themselves lacking in the depth of visibility into applications. Although agent-based monitoring provides information about individual systems and agentless monitoring fills out the picture with network statistics, a much deeper level of understanding is still necessary.

That "deeper level of understanding" arrives with a type of monitoring that digs past aggregate network statistics to peer into the individual transactions themselves between elements of a system. By looking at individual transactions that occur between system elements, it is possible to look for areas where code performance or inter-server communication is a fault.

To give you an idea of how transactions work, think about the last time you clicked on a link for an image in your favorite browser. Clicking that link directed your browser to request the download of the image. In a few seconds or less, that image was later rendered on your browser for you to view.

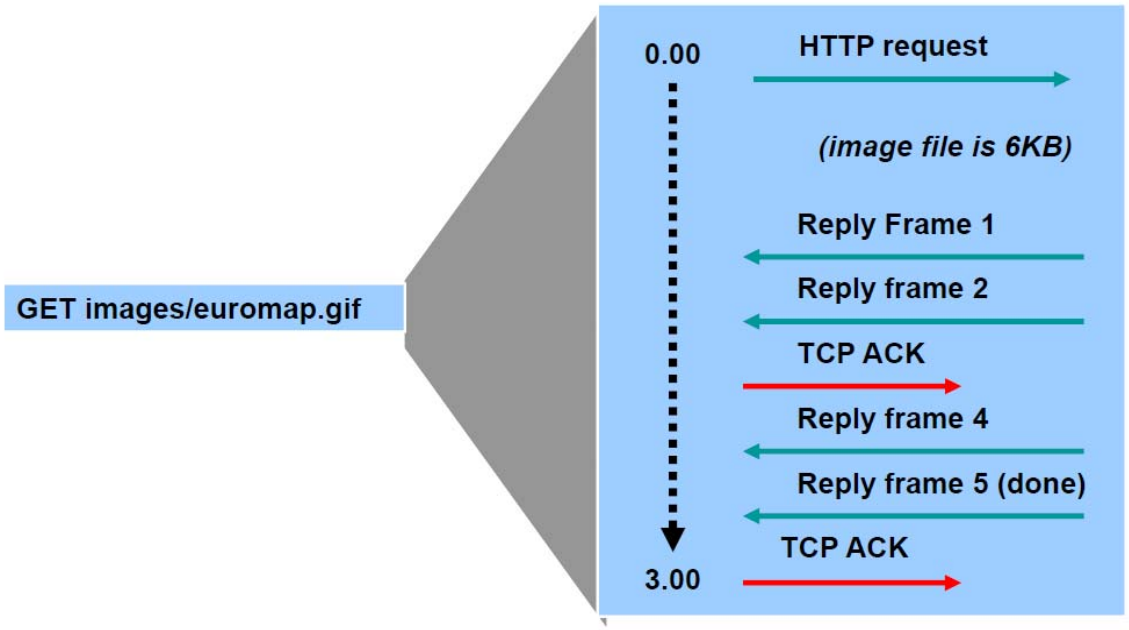

Figure 3.8: Multiple conversations ("transactions") must occur for a single image to be downloaded from server to browser.

But what goes on in the background when such a request is made? What kinds of conversations are required between your local computer and the remote server for that image to successfully make its way across the Internet to your laptop? In actuality, the conversation between client and server can be amazingly complex. Figure 3.8 shows an example of the communications that must occur for the image *euromap.gif* to be successfully downloaded off the Internet. Requests, replies, and acknowledgements are all required steps for what seems simple on the surface.

Transaction Monitoring in Action

In Figure 3.8, you can easily see the layers of complexity involved with downloading just a single image. Now expand that necessity across all the servers—database, business logic, mainframe, and presentation—that make up the application or service you want to monitor. A simple user request to add an item to their shopping cart can immediately launch a chain of events across multiple servers in the environment:

- A Web server queries a business logic server to process the request.
- A business logic server queries a mainframe for product inventory and price.
- A mainframe verifies product inventory and provides a response to a business logic server.
- A business logic server increments that user's shopping cart token by one.
- A business logic server instructs a Web server to refresh the user's profile with the updated count.
- A Web server then refreshes the page, reporting the successful addition back to the user.

Consolidating all those transactions into a visualization that makes sense for the administrator is no easy task. The system must collect the right information; it must also present that information in a way that is digestible for its user. Effective APM solutions enable multiple mechanisms for visualizing the communication between components, including Thread Analysis Views like what is shown in Figure 3.9 and Conversation Maps similar to Figure 3.10.

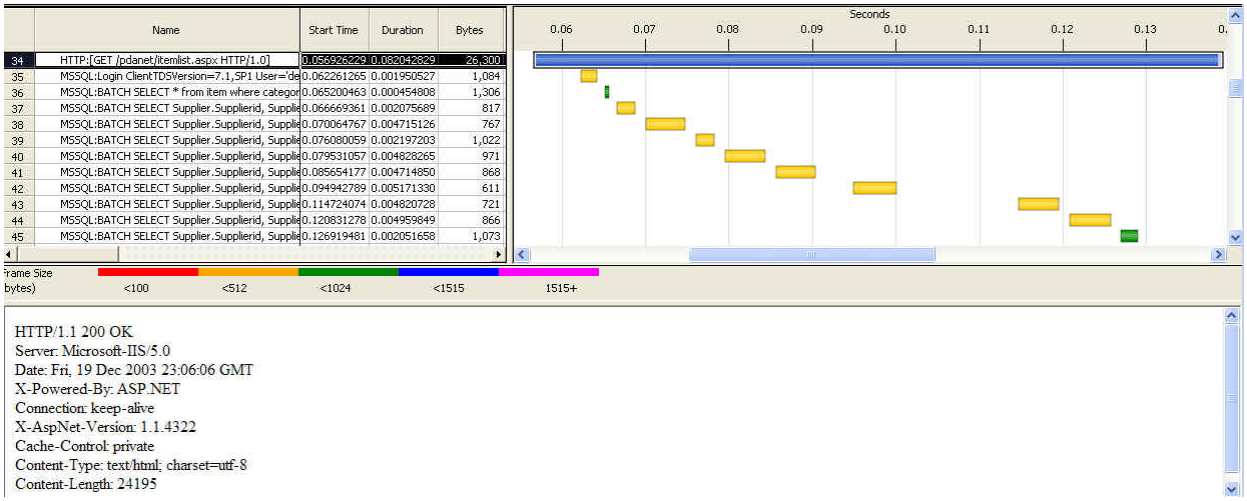

Figure 3.9: A Thread Analysis View provides a look at transaction details by time.

The information in a Thread Analysis View is gathered through the analysis of particular types of traffic occurring between identified application components. In Figure 3.9, an HTTP GET command is being analyzed along with backend SQL queries related to the request. Identifying the source and destination of the transaction along with its payload and description assists the troubleshooting administrator with deconstructing the transaction into its disparate components. This process is similar in function to the analysis of a network trace done with network management tools. However, unlike the network trace's focus on individual packets, here the analysis is elevated to the level of the transaction.

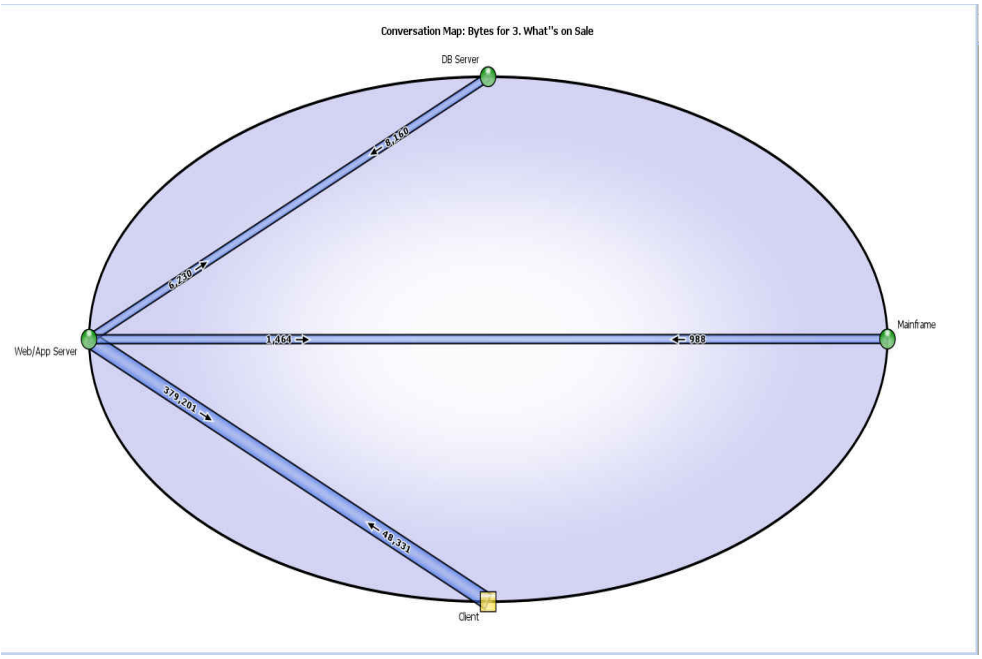

Figure 3.10: A Conversation Map illuminates which components are talking with whom.

Elevating the analysis even further creates a Conversation Map view. This high-level view assists the administrator with a look at the components involved in a transaction as well as characteristics about the communication. This information is useful for identifying which participants might be the cause of a performance issue or other problem.

These graphics are obviously only a small portion of the visualizations that can be created through the use of an effective APM solution. Chapter 7 will focus exclusively on visualizations like these, while Chapter 8 will highlight a specific troubleshooting example through the use of an extended example.

Application Runtime Analysis

Yet another area of integration relates to the applications themselves. Most business applications are comprised of numerous applications, code frameworks, and middleware elements that work in concert to provide a service. These separate but integrated components are necessary as each provides some function for the end solution.

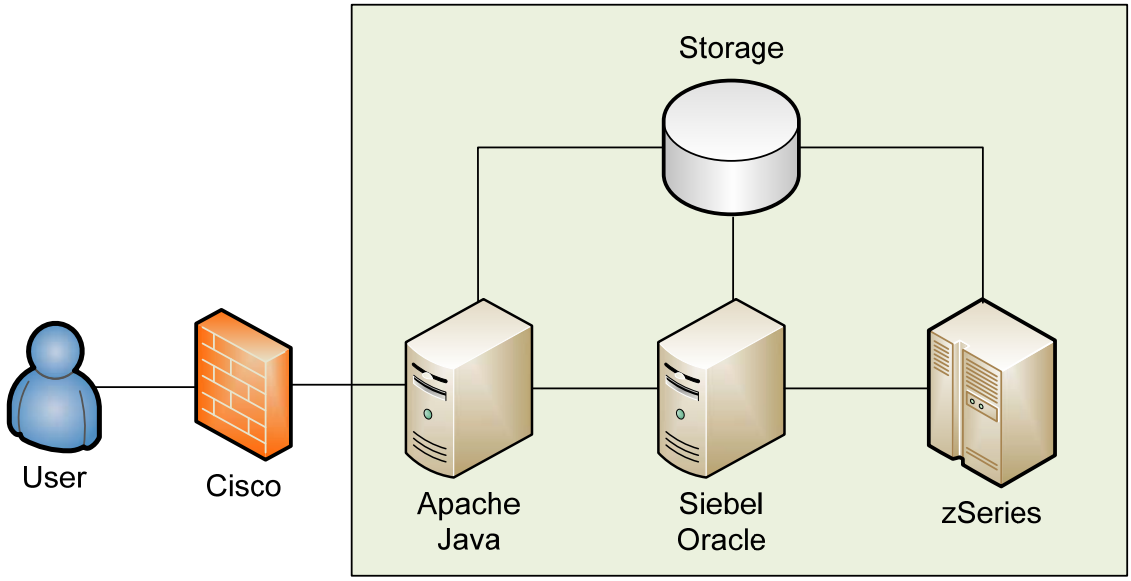

Figure 3.11: Product-specific hooks provide deep insight into their behaviors.

Let's re-imagine the simplistic environment once again, this time attaching some well-known products to the otherwise generalized terms "Web Server," "Application Server," and so on. Figure 3.11 shows this environment once again, showing a Cisco firewall, an Apache Web server running custom Java code, an Oracle database, and Siebel middleware, all connecting back to a zSeries mainframe. Although the actual product names themselves are unimportant for this discussion, the fact that shrink-wrapped products are components of this environment is.

APM's transaction-level monitoring enables the capacity to peer into the individual conversations that occur between servers and applications in your environment. Yet today's enterprise applications themselves also support pluggable mechanisms for gathering instrumentation data directly from the application itself. This instrumentation data can provide additional insight into the inner workings of the applications in your infrastructure.

Consider the situation where a custom-built Web site is created atop an Apache Web engine and built in part using the Java language. In this case, determining the inner performance characteristics of the Web server and language might be best served by querying directly to Apache's and Java's internal metrics frameworks. This internal information can be merged with transaction statistics to gain an understanding of where processing delay is occurring—at the client, on the server, or within the network.

End User Experience

Concluding this discussion on APM's monitoring integrations is a discussion on monitoring's "last mile." The behaviors of your application that are seen at the client itself are in many ways the most important facet of any APM solution. Think for a minute about this chapter's evolving conversation on monitoring. When looking at a large-scale business application, it is, for example, possible to

- Measure the memory use on a server using on-board agents
- Measure the transaction rate between middleware systems and databases leveraging agentless network monitoring
- Measure the processing rate within a language framework or application

However, by itself, none of this information directly gives you any information about what the user experiences when they click the "Add to Cart" button on your Web site.

It can be argued that End User Experience (EUE) monitoring encompasses some of today's most advanced monitoring technologies. It is, after all, one of the most recent of the available enterprise monitoring approaches. EUE monitoring provides its value by measuring the performance of the application from the perspective of its ultimate customer—the end user.

EUE functions in three very different but very important ways. The first of these is through the introduction of client-based monitoring at the user's location itself. This can occur through an agent's installation to an end user's location or through the use of special probes. By co-locating an agent with the end user, that agent is enabled to monitor the known behaviors of your system. It can then look for situations in which end-state performance back to the user has degraded past acceptable thresholds. By locating monitoring agents at the client itself, the individual transactions associated with end user behaviors can be mapped out and timed to validate that your end users are experiencing the right level of service.

A second way to measure the end user performance of your applications is through the use of automated "robots." Also located in areas where end users make use of your application, these robots run a set of predefined scripts against your application. These scripts leverage synthetic and actual transactions that are very similar to the types of actions a typical user would perform against the system. For example, if users click through a Web site, attempt to add items to a shopping cart, and ultimately check out, these types of actions should be simulated by the automation robot.

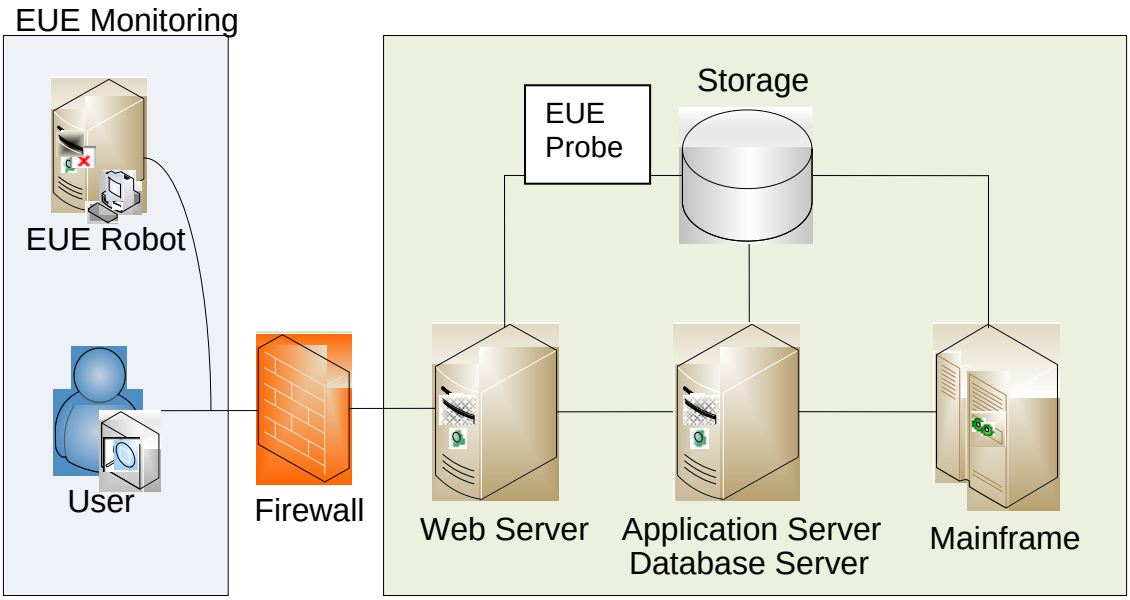

Figure 3.12: EUE integrations can be co-located with the users themselves or run through robots for consistent automation.

Since these actions and their scripts are well-defined, the timing associated with their processing is also a known quantity. As the robot runs the same scripts over and over, it is possible to quickly determine when service quality diminishes for the end user. EUE is a powerful component of APM monitoring, one that is arguably its greatest value proposition for your business applications.

EUE on the Enterprise WAN

Although EUE is an obvious play for Internet-based applications, a similar set of benefits can be found with enterprise applications that operate across a Wide-Area Network (WAN). Similar in function but different in scope to the agentless approach are the enterprise WAN monitoring functions that EUE provides. These functions are more geographic in scope rather than operational; however, they are no less useful for organizations that span multiple sites, countries, and continents.

Incorporating EUE's into the enterprise WAN may require the installation of agents or robots across many or all the individual sites within the WAN network. By installing these components to multiple locations in the environment, the end users' perspective can be measured based on geographic location and any network behaviors that are experienced in that locale.

For example, consider the situation where a business application is homed in Denver, Colorado but is used by employees through the United States, EMEA, and Australia. That same application might function with no problems for the users in the United States. The latency found in those connections might mean that even low-bandwidth connections support the application's use with no issue to the end user.

However, EMEA and Australia users might see a different situation entirely. Due to the realities of physics, even the fastest connection between the United States and other continents adds a specific quantity of network latency. If your business application is latency intolerant but bandwidth insensitive, the users in those locations could be on the fastest connection possible but still see a low-quality experience with your application.

In short, EUE's *ability to quantify the user's experience* is crucial to maintaining your customer satisfaction.

APM = ∑ the History of Monitoring

This chapter began by explaining how a history of monitoring is necessary to truly understand APM. This is the case because APM's expansive monitoring capabilities encompass the summation of that history. Businesses and their network environments have evolved their network, system, and application monitoring over the years to include integrations at virtually every layer of the application infrastructure. Monitoring server and network metrics are augmented through transaction data. Transaction data is further augmented by a focus on the user's perspective. Centralizing that data into a unified solution across every integration point brings power to what would otherwise be a collection of point systems.

With this information associated with monitoring's potential, the next topic really relates to how it can be implemented into your existing network environment. Chapter 4 discusses the processes and practices associated with integrating APM into your existing application infrastructure. Tiering its monitoring integrations across users, applications, databases, and mainframes requires multiple integrations. Chapter 4 will assist you with understanding how to accomplish those tasks correctly.