Packeteer Technical White Paper Series

Response-Time Technology

May 2002

Packeteer, Inc.
10495 N. De Anza Blvd.
Cupertino, CA 95014
408.873.4400
info@packeteer.com
www.packeteer.com

Table of Contents

Response-Time Technology

PacketShaper's response-time management (RTM) facility quantifies what has traditionally been subjective, anecdotal information. RTM tries to measure the user experience.

PacketShaper's position in the network — monitoring all the traffic that passes — gives it a unique opportunity to provide accurate response-time measurements at a very low cost. Because it already sees every packet, PacketShaper can easily calculate the time traffic spends traveling between a client and a server, the time used by the server itself, and the time spent on either side of PacketShaper itself. Rather than gathering or collecting response data, PacketShaper notes response times as traffic passes. This simple approach provides rich data without network impact or overhead.

This white paper summarizes RTM features and then delves into technical detail about how PacketShaper calculates response-time measurements.

RTM Features

PacketShaper RTM offers the following features:

- Tracks delay statistics for any TCP-based traffic class. For example, you can measure response times for applications such as Oracle, SAP, Citrix-based PeopleSoft, one or more applications using an MPLS path, the traffic to or from individual hosts, subnets, particular web pages of your choice, and much more.
- Breaks each response-time measurement into network delay (time spent in transit) and server delay (time the server used to process a request). Several other response-time metrics are also available and are detailed later.
- Identifies the users and servers with the slowest performance, called Worst Clients and Worst Servers.
- Sets acceptability standards and tracks whether performance adheres to the standards. You can set the speed that divides good response from bad (e.g. 900 ms), and you can set the percentage of transactions that should be within your performance goal (95 percent, for example).
- Offers current and historical performance data in tables, graphs, in a MIB (management information base), via an API (application programming interface), or as raw data.

PacketShaper's Response-Time Advantages

PacketShaper's strengths in the response-time arena are based significantly on what it does *not* do and from the problems it does *not* impose. PacketShaper avoids common pitfalls, including:

- **Application modifications**

  PacketShaper does not require software wrappers around measured applications or the addition of API calls.

- **Desktop and server changes**

  PacketShaper does not need anything loaded on client desktops or any server.

- **Artificial traffic overhead**

  PacketShaper does not create extraneous application requests merely to time their responses, and PacketShaper does not issue pings. It can issue synthetic transactions if requested, but calculates all response-time metrics without resorting to artificial traffic.

- **Router reconfiguration or topology changes**

  PacketShaper requires no changes to router configuration, protocols, servers, desktops, or topology.

- **Location restrictions**

  PacketShaper measures performance anywhere on the network, as long as it sees the traffic it is measuring. It can sit on the client side or on the server side of the topology. If the Internet separates the client and server, it does not matter on which side PacketShaper is deployed.

- **Data collection overhead**

  PacketShaper does not load the network with constant statistics downloads.

Response-Time Metrics and Descriptions

PacketShaper offers a variety of response-time metrics. By using the metrics appropriately, you can baseline performance, spot performance problems, and identify their location. In addition, you can enter response-time commitments in PacketShaper, then measure compliance and request notification if compliance is not up to par.

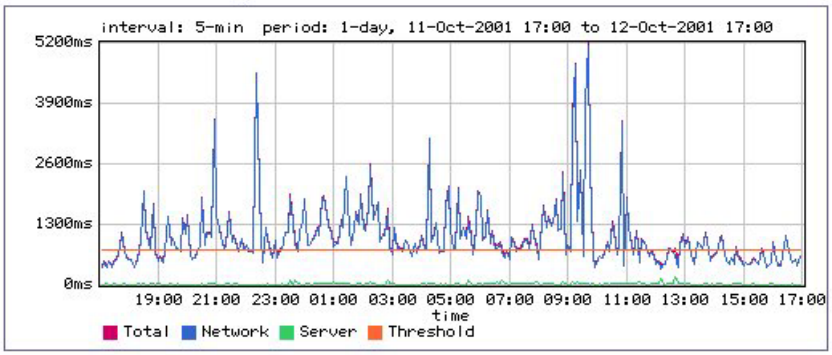
Total Delay

The number of milliseconds beginning with a client's request and ending upon receipt of the response. This is what most people mean when they say "response time," corresponding to the end user's view of the time it takes for response after initiating some action.

Network Delay

The number of milliseconds spent in transit when a client and server exchange data. If a transaction requires a large quantity of data to be transferred, it is divided and sent in multiple packets. Network delay includes the transit time for all packets involved in a request-response transaction. The amount of time the server uses for processing a request is not included.

Transaction Delay



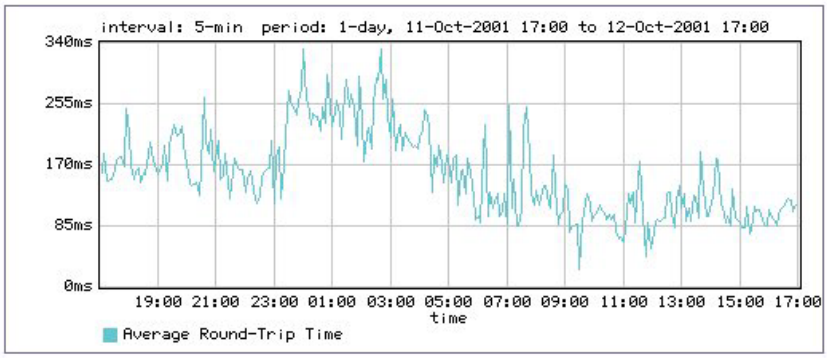
Server Delay

The number of milliseconds the server uses to process a client's request after it receives all required data. The server delay is the time after the server receives the last request packet and before it sends the first packet of response (not the receipt acknowledgement, but actual content). This is the time the server takes to process the client's request.

Normalized Network Delay

The number of milliseconds per kilobyte spent in transit when a client and server exchange data. If a transaction requires a large quantity of data to be transferred, it is divided and sent in multiple packets. Because network delay increases as transaction size increases, it can be misleading when comparing times. Normalized network delay eliminates size as a factor so that comparisons are more reasonable.

Packet Round-Trip Time



Round Trip Time (RTT)

Description: The number of milliseconds spent in transit when a client and server exchange one small packet. Even if a transaction's data is split into multiple packets, RTT includes only one round trip of a single packet between client and server.

The Good and the Bad

Although delay statistics are useful, it's even better to have a thumbs-up or thumbs-down performance indicator. After all, what does a metric like 600 milliseconds really mean? Is it good or is it bad?

PacketShaper provides the ability to set acceptability standards and track whether performance adheres to the standards. You can set the speed that divides good response from bad (500 ms, for example), and you can set the percentage of transactions that should be within your performance goal (95 percent, for example).

Packet Exchange Time (PET)

The number of milliseconds between a packet's departure from PacketShaper and receipt of the corresponding acknowledgement. This metric reflects only the delay for the network on one side of PacketShaper.

PET is useful when PacketShaper sits at a service boundary, where each of two parties is responsible for a portion of the network. PET measures each party's delay to help determine appropriate responsibility for slow-downs.

Synthetic Transactions

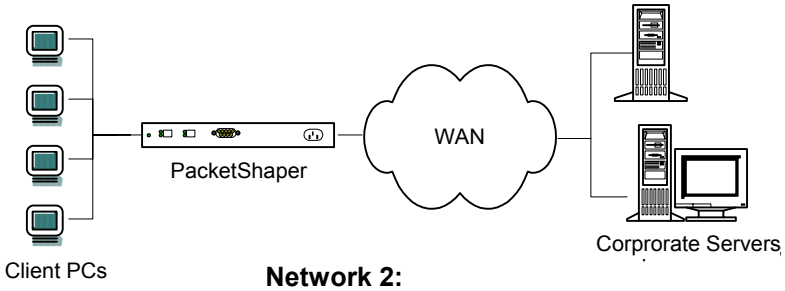
PacketShaper can, at your discretion, initiate web or other TCP transactions at periodic intervals to verify the availability of critical hosts. This activity is similar to scheduled pings or SNMP polls, but with these important advantages:

- PacketShaper's detailed analyses of transaction behavior and response times apply to synthetic transactions, rendering the ability to profile network and host behavior over time. This information is much more helpful than knowing if a device simply responds to pings or not.
- Because PacketShaper sits at the network edge, polls are local, consume less bandwidth, and can therefore be more frequent.
- Synthetic transactions can determine if a service or application is running, not just if a server is responding. They provide a more sophisticated assessment of "availability."
- Distributed PacketShapers can serve as local clearinghouses for availability information, forwarding situations of interest or concern to central locations via email, SNMP traps, or syslog messages. The need for long-distance polling by central management platforms is eliminated.

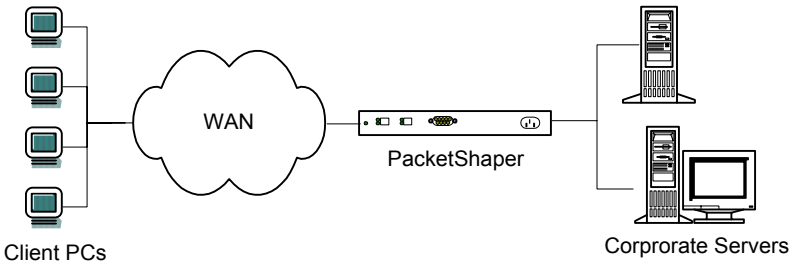
Working in a Variety of Topologies

PacketShaper easily fits with your topology, rendering response-time measurements whether it sits close to clients or to servers and whether or not there is a WAN link in the picture.

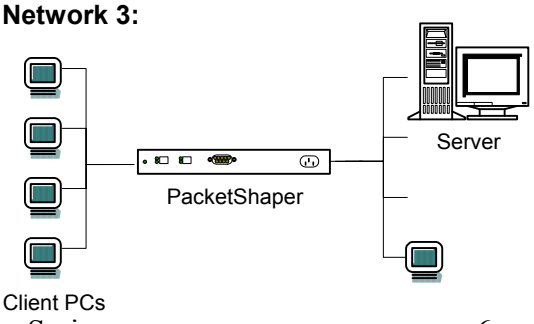
Network 1:



Network 2:



Network 3:

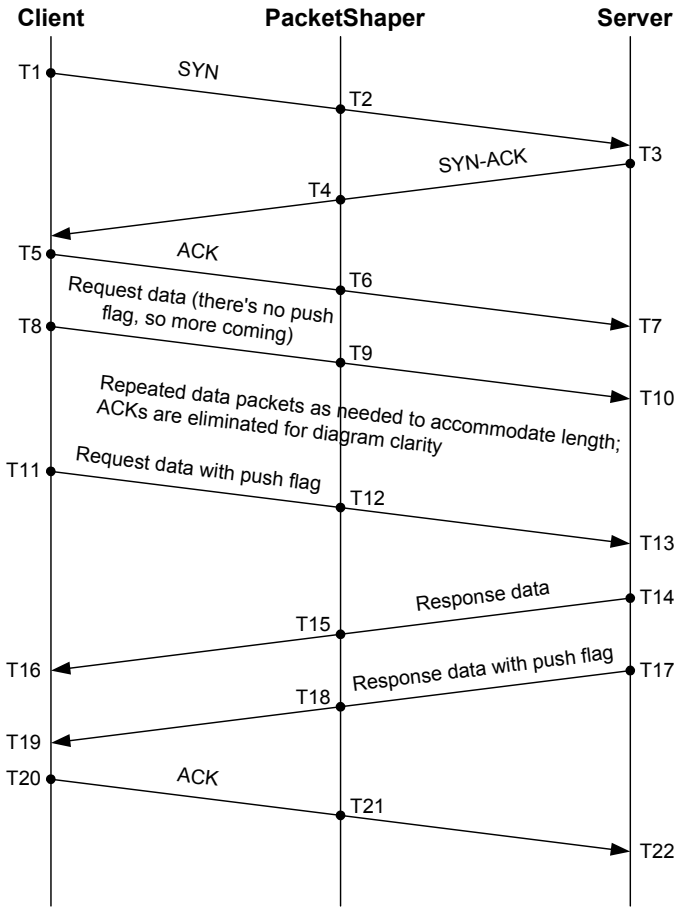


The topology diagrams show three typical arrangements. The first shows a client-side PacketShaper, the second shows a server-side PacketShaper, and the last shows PacketShaper in a LAN environment. Note that PacketShaper must see the traffic it analyzes. In the third diagram, if the client that sits by itself exchanges information with the server, its traffic does not go through the PacketShaper and therefore would have no associated response times.

Calculating Delays

PacketShaper tracks the course of a client-server transaction and uses information about a TCP connection to differentiate one portion of the exchange from another. The following diagram helps illustrate PacketShaper's insight into a connection's components.

The illustration is a standard TCP diagram showing the course of a network transaction over time. Arrows indicate packets traveling the network between client and server. Time increases as you descend, with successive events' times noted as TN, T1 representing the first event and T22, the last.



A client initiates a server connection with a SYN at time T1. PacketShaper notes the SYN at time T2 and passes it along to the server. The server responds with a SYN-ACK at time T3. PacketShaper notes the SYN-ACK at time T4, passing it along as usual.

TCP stacks usually respond with a SYN-ACK very rapidly, within the kernel and with no context switches. The SYN-ACK follows the SYN almost immediately. Therefore time T4 minus time T2 results in an accurate measure of the round-trip network delay between PacketShaper and the server. This interchange produces the first quantity, the server transit delay (STD):

```
STD = T4 – T2
```

The client receives the SYN-ACK and issues the final ACK of the three-way handshake at time T5. PacketShaper notes the ACK at time T6, passing it along to the server.

It is reasonable to assume that no processing transpires between the client's receipt of the SYN-ACK and its own corresponding ACK at time T5. Time T6 minus time T4 yields an accurate

measure of the round-trip network delay between the client and PacketShaper. The client transit delay (CTD):

$$CTD = T6 - T4$$

Putting together the server transit delay and the client transit delay yields the total delay between the client and the server for a single round trip.

$$RTT\ (Round\text{-}Trip\ Time) = STD + CTD$$

Determining the Server Delay

The client initiates its request at time T8, arriving at the PacketShaper at time T9. For large requests, the request is divided into multiple packets. The TCP diagram eliminates the server's corresponding ACKs to simplify the picture, because these ACKs are not material to PacketShaper's calculations. The last request packet, sent at time T11, has its Push Flag set to one indicating it is the final packet. PacketShaper notes the time of this last request packet at T12.

After the last request packet arrives at the server at time T13, the server assembles the request, conducts whatever processing is required for the request, and assembles its response. The server sends the first packet (of potentially several response packets) at time T14. Time T14 minus time T13 is the actual server-processing time required for the request, but these times are not visible to PacketShaper.

PacketShaper knows that the server's processing time occurred after it saw the last request packet and before it saw the first response packet (time T15 minus time T12). Additionally, it knows that another component of this interval was the transit time from PacketShaper to the server and back again. Conveniently, it already has that figure — it's the server transit delay (STD). In addition, there is a small amount of time spent serializing the bits in the response packet and preparing them for their bit stream. This time was not included in the original server transit delay because the SYN and ACK packets are extremely small. PacketShaper knows the size of the packet, calculates this preparation time accordingly ($\Delta 1$), and adds it to the STD before subtracting the sum from the time difference.

$$Server\ Delay = (T15 - T12) - (STD + \Delta 1)$$

Determining the Total Delay

The termination of a transaction is key to calculating the total delay; however, it's not always obvious when a transaction ends. The combination of a Push flag from the server and its corresponding ACK from the client frequently signal the end. But long transactions often insert Push flags throughout the transaction.

In addition to monitoring Push flags, PacketShaper uses a timer to track transactions and uses the following rules:

- If a Push flag seems to indicate a transaction's end, but the server continues sending more data, the timer continues to advance.
- If the client sends a new request, PacketShaper ends the last transaction and records the last time noted.

- If there is no activity from either the server or the client, PacketShaper considers the transaction complete and records the last time noted.
- When the connection ends, PacketShaper sees the FIN and records the last time noted.

Using these techniques, PacketShaper notes the last response packet at time T18, makes sure that it saw all required ACKs for the request packets, and verifies that the last response packet indeed represented the end of the transaction.

After the client receives the final response packet at time T19, it sends an ACK. The ACK reaches PacketShaper at time T21. The client's perspective of response time starts with sending the first request packet (T8) and ends with receipt of the final response packet (T20). PacketShaper sees that interval as time T9 until time T21. Although this is a close estimate of the client's view, it's missing some extra preparation time for serializing the first request packet, assuming it is larger than the final ACK. Because PacketShaper knows the packet-size difference, it can calculate this small discrepancy ($\Delta 2$).

$$\text{Total delay} = (T21 - T9) + \Delta 2$$

Once PacketShaper has the server delay and the total delay, it can calculate the amount of time the transaction spent in transit.

$$\text{Network delay} = (\text{Total delay}) - (\text{Server delay})$$

Whereas the RTT represents the transit time for just one round trip, the network delay reflects all transit time for the transaction. If the transaction's data is large, multiple packets need to make their way to and from the server. Only the network delay reflects this overhead. The network delay is not necessarily an even multiple of the RTT because multiple packets are not sent consecutively but tend to overlap to varying degrees. In addition, because network and total delay are products of transaction size, ping times and RTM measurements are not comparable.

As you can see, PacketShaper uses its intermediary position to make time and size observations during a transaction. Then it incorporates TCP basics to render accurate performance figures.