

# ShapeShifter

Traffic Shaping Engine  
Beta TSE 1

Network Manager's Guide

July 2000



## Read Me First!

### Important Beta Caveats

- The Beta version of the TSE has limitations you should be aware of prior to loading it on your server. *Please read the following carefully* prior to installing the software on your server or loading it from the server console.

***The Beta TSE has not been tested under NetWare SFT III.*** Under **no** circumstances should the Beta TSE be used on NetWare SFT III platforms. Doing so may cause the server to AbEnd.

***The Beta TSE has not been fully tested with NetWare SMP.*** While the initial tests look good, there is not enough experience running the TSE on SMP platforms. The TSE has been tested on a couple SMP configurations, but this testing is by no means complete. If you have a test server with SMP, the results of your tests are very important. Please provide feedback about the TSE's stability on these systems.

***The Beta TSE is incompatible with non-Ethernet topologies.*** Under **no** circumstances should the Beta TSE be bound to FDDI, Token Ring, or ATM interfaces. Doing so may cause the server to AbEnd. The TSE *can* be used on any Ethernet, Fast Ethernet, or Gigabit Ethernet NIC. Support for all major topologies will be available in future releases.

***Exercise caution when enabling debug logging.*** The logging facilities of the TSE are still geared for debugging and can produce extremely voluminous log files. While the default is minimal logging, console commands can enable detailed logging of each frame, queuing, rule processing, and hashing events.

***Normal operation of the TSE can cause disruptions.*** Like any traffic management system, the TSE limits and discards excessive network traffic. It is possible to configure the TSE in such a way as to interfere with normal or desirable network traffic. Monitor the TSE carefully when implementing a new configuration. Be ready to unload the TSE in case your configuration does not work as expected.

**Use caution when using the Configuration Utility.** When the TSE is running, changes made through the configuration tool are *automatically loaded* by the TSE. Back up the configuration file prior to making configuration changes.

***This version of the TSE is a Beta release.*** The Beta TSE is the first public release of the TSE binaries. It is *likely* that the TSE contains program bugs that could cause network disruptions or ABEND. The goal of the Beta release is to identify such issues and resolve them prior to a production release. ***You use the Beta TSE at your own risk. See the License Agreement for details.*** This release is not recommended for use on a production server. Do so only be after trouble free experience running the TSE on a similar non-production server. Even then only when the benefits of using the Beta TSE outweigh the risks. **Risks include network disruptions, server AbEnd, and data loss, possibly extending to other network attached systems.**

## Introduction

The ShapeShifter Traffic Shaping Engine ( TSE ) is the first of two components that constitute a complete bandwidth management system for NetWare servers. The TSE is a NetWare Loadable Module ( NLM ) for NetWare 4.x and 5.x servers that adds basic traffic shaping capabilities to NetWare and applications that run under Netware.

The Beta TSE offers the ability to rate limit any desired traffic. Policies can be applied to connection oriented traffic to rate limit conversations between hosts. Threshold driven rate limiting and bandwidth quotas provide an effective means of managing Internet and LAN bandwidth.

## Purpose of the TSE

The TSE adds bandwidth management functionality to the server. The TSE regulates the flow of network traffic in and out of the server's network interfaces. The Beta TSE supports static and dynamic configurations for identifying and managing traffic. The configuration tool provided manages the static configuration. Dynamic configurations allow instant changes to bandwidth policies in reaction to external events. Add-on modules will extend the TSE's functionality through dynamic configurations. The NDS Manageability Module will be implemented in this way.<sup>1</sup>

Static configurations control rule based identification and grouping of traffic and application of data rates to the groups. The Beta TSE is unaware of NDS users, groups, organizational units, etc. Despite this, blanket policies regarding bandwidth quotas and data rates can be created using static configurations. Effective bandwidth policies can be enforced with simple configurations.

---

<sup>1</sup> The NDS Manageability Module ( which is still under development ) allows the TSE to manage bandwidth for User, Group, and Container objects. The TSEs running on several servers can be clustered to share bandwidth information providing fault tolerance and managing multiple entry points into the LAN.

## Structure of the TSE

The TSE is a NLM that runs on NetWare 4.xx and 5.x servers. To NetWare, the TSE acts like a protocol module, much like TCPIP. Once loaded, SHAPER.NLM implements the SHAPER “protocol.” The SHAPER protocol is bound to logical boards by using the standard BIND and UNBIND commands at the server console.

Once bound to an interface, the NLM intercepts all network traffic entering and leaving the logical board. The TSE can bind to as many as 4 logical boards selected from any Ethernet NICs present. This is a Beta limitation.

Traffic passing through the TSE is processed by a rule base filtering mechanism. The primary purpose of a rule is to identify groups of traffic to be managed. Rules can drop, forward, or apply a data rate to the frames selected. Rules are used to forward traffic the TSE is to ignore. Traffic to be managed is directed to an Address Mask or QoS Node. The TSE *can* be used to filter selected traffic, but was not specially *designed* to be a filtering mechanism. Robust filtering capabilities are provided by NetWare's IPFLT or IPXFLT utilities.

The TSE's queuing mechanism applies the data rate. The queue smoothes out bursts of traffic associated with sliding window protocols like SPX and TCP. It also provides a means of releasing these bursts over longer periods of time, slowing the speed of the connection to any arbitrary rate. The queue allows data rates as low as 2K per second, effectively the speed of a 14.4K modem, to be applied without interfering with the windowing protocols or generating excessive retries. Data rates can be set arbitrarily high, but the practical limit, per connection or rule defined group, is 10MB per second, equivalent to a fully saturated Fast Ethernet connection.

The accuracy of the rate depends on the ability of the queue to accurately release queued frames in the proper amount of time. There will always be a level of error, based on server load, that slightly retards a given frame. This delay is usually of the order of 50µs. When the data rate is low, for example 100K per second, the average delay is about 10000µs. A 50µs variance is not a “real” problem. As the average delay decreases to a level comparable to the variance, the effective throughput falls below the data rate

desired. A data rate of 1000K per second will typically result in 920K of throughput. Data rates below 1MB per second are accurate to within  $\pm 10\%$ .<sup>2</sup>

## Rules

The TSE intercepts received and transmitted frames passing through the interfaces to which the TSE is bound. Once the TSE has the frame, it executes the first rule associated with the half-bind<sup>3</sup> the frame is passing through. The incoming and outgoing traffic can be managed separately, each with a different initial rule. Each rule acts as an “If *condition* Then... *action* Else... *action*” unit. Rules either conditionally branch to other rules or perform some action against the frame being tested. Actions include forwarding, dropping, or managing the frame.

Various tests can be performed against the frame's contents as well as against a NetWare data structure called an Event Control Block, the ECB. A single rule can test up to 16 contiguous bytes of data against the pattern supplied. A combination of offset type and offset value specifies the part of the frame or ECB to test. A mask identifies “don't care” bits in the pattern. Masks direct the TSE to test a specific combination of individual bits or bytes within the 16 byte span.<sup>4</sup> Each comparison takes the same amount of time to execute, irrespective of the length of the pattern. Masks can be used to perform complicated comparisons that might normally require several individual tests, and so offer improved efficiency.

The pattern and mask are used in standard mathematical style comparisons:  $>$ ,  $<$ ,  $=$ , and their negations,  $\leq$ ,  $\geq$ ,  $\neq$ . The  $\$$  and  $\&$  operations are not yet implemented, but will eventually perform sub-string searches. For comparisons of magnitude, the most significant bytes are to the left, i.e. must come first in the frame. The magnitude operators:  $>$ ,  $<$ ,  $\leq$ , and  $\geq$  will perform as expected on most frame data. These operators, for example, could be used to identify any arbitrary range of ports or IP addresses.

---

<sup>2</sup> The effect is only a difference between the *desired* and *actual* data rate and does not reflect a decrease in efficiency of the TSE or data loss. These deficiencies will be addressed in future releases.

<sup>3</sup> A half-bind manages either the incoming or outgoing frames for a particular binding of the TSE to a logical board.

<sup>4</sup> This is particularly valuable when testing for option bits in IP frames, within TCP headers, or when looking for specific combinations of address and port.

## Constructing Comparisons – An Example

Creating effective rules and comparisons can be a difficult task. The best way to start is to identify the objective and translate that objective in to a series of simple tests that implements the criteria.

Suppose the goal is to apply a data rate to all web traffic entering our network, e.g. the *external content* being browsed by our *internal browsers*. The criteria would be:

1. Frame is inbound IP traffic
2. Frame is destined for our network
3. Frame is a TCP frame
4. Frame is from port 80 ( HTTP )

The first hurdle is to identify an IP frame. This can be accomplished in a variety of ways. For this example we will use the ECB's Protocol ID field. Ethernet encapsulated IP frames have a Protocol ID of 00.00.00.00.08.00. This is found at offset 16 into the ECB. Frames matching this rule are IP frames.<sup>5</sup> Now we need to perform further tests to ensure the IP frame is one we want to manage.

The TSE configuration utility accepts the pattern and mask data in hexadecimal form. So all the decimal numbers need to be expressed in hexadecimal. If our IP network is 10.12.22.0/24<sup>6</sup>, any address in our domain always begins with 10.12.22. This is 0A.0C.16 in hex. TCP port numbers are 16-bit integers, 80 is 00.50 in hexadecimal. TCP frames are type 06 IP frames. After examining the layout of an IP and TCP headers, as show in the Protocol Reference appendix, you assemble these facts:

Offset 9 into IP Header

Type	Checksum	Source IP Address				Destination IP Address			
06						0A	0C	16	

TCP Header

Source Port	
00	50

<sup>5</sup> See Case 2 in the Sample Applications section of this manual.

<sup>6</sup> A Class C subnet of a Class A network. The /24 indicates that the first 24 bits are network bits. The network mask would be FF.FF.FF.00 or 255.255.255.0.

Any IP frames matching the bytes shown above meet the criteria! The mask tells the TSE which bytes or bits you “don’t care” about, those left blank above. The data and mask you would supply the configuration tool would be:

```
Data: 06.00.00.00.00.00.00.00.0A.0C.16.00.00.50
Mask: FF.00.00.00.00.00.00.00.FF.FF.FF.00.FF.FF
```

This single comparison tests for all three criteria at once. Only 13 of the 16 byte span was used.<sup>7</sup> You could have created 3 separate rules, each testing for an individual element of the criteria, but that would be less efficient. The destination network address is selected using a portion of the mask. This is possible since all the destination addresses share a common bit signature, namely 0A.0C.16. This same principle can be used to identify port number or frame length ranges. The masking feature conveniently handles sub-net masks, i.e. part of the mask *is* the sub-net mask, as indicated by the underlining shown above.

Keep in mind that each rule can be executed several *thousand* times a second. Rules need to be as efficient as possible. Each rule executes in 100 to 500 CPU cycles.<sup>8</sup> It takes a lot of rules to significantly impact performance. But do not go overboard. Poor rule performance can be resolved by analysis of branching statistics, available through console commands. Rearranging rules so the most selective execute first will greatly decrease rule overhead.

The TSE also offers access to the ECB, which is an internal NetWare data structure. An ECB is associated with each frame as it travels through the server. Choosing the ECB offset type makes the TSE perform your comparison against the ECB data structure rather than frame data. The ECB contains information about the frame type and protocol ID which can be very useful in making rules differentiate between frames with similar looking data and no easily constructed litmus tests for frame type and contents. The ECB also contains a lot of information you do not care about – so be cautious.

---

<sup>7</sup> As can be seen in this example, it is possible to test for the type of IP frame, source and destination IP address / network, source and destination port, all in a single comparison operation.

<sup>8</sup> A CPU cycle is roughly analogous to a clock cycle. A decent Pentium based server can execute several hundred thousand rule comparisons per second. The TSE has executed 250,000 comparisons per second on Pentium/133 based systems with minimal overhead.

## Sent vs. Received Frames

While the TSE sees raw frames *received* by the server, it does not see the raw frames *sent* by the server. At the point where the TSE is passed outgoing frames, the server has yet to construct the actual frame headers. The MLID does this just prior to the server transmitting the frame on the wire. The TSE must produce a facsimile of the missing frame headers when performing comparisons against outgoing frames. This process requires that the TSE be frame and protocol aware. The TSE is aware of various Ethernet frame types and the IP and IPX protocols. For these frames, the difference between incoming and outgoing frames is transparent and requires no difference in the rules.

There is only an issue if the comparison, specified by the rule, requires access to the frame headers for encapsulations that the TSE is unfamiliar.<sup>9</sup> The construction of the missing header information adds some overhead to comparisons on outgoing frames. This overhead can be avoided by using fragment or protocol offsets. These offset types do not need the frame header to be constructed. Future releases will include support for most common frame types and protocols as well as for FDDI, Token Ring, and ATM interfaces.<sup>10</sup>

## Rule Actions

Depending on the outcome of the comparison, a rule performs the action specified in the Then or Else clause. Actions come in two flavors: rule execution and frame management. Rule execution actions determine which rules are tested next. A rule can skip to the next rule or jump to a particular rule.<sup>11</sup> The second action type, frame management actions, control the processing of the frame. The rule can forward, drop,

---

<sup>9</sup> For example, the TSE is not “aware” of the AppleTalk protocol. To manage AppleTalk you would need to construct rules that identify AppleTalk frames, perhaps by using the ECB’s Protocol ID field. Only the FRAGS offset type would work consistently between send and received frames. To construct rules to test the frame headers and envelope, you would need different rules for incoming and outgoing frames.

<sup>10</sup> Novell’s Topology Specific Modules or TSM is designed to provide a level of abstraction for dealing with encapsulations. However, there is no public API to accessing TSM features. Developers are left with the task of writing their own code to handle this. This is true of the TSE, which translates to lagging support for odd protocols and waning topologies.

<sup>11</sup> The Configuration Tool shows “sequence” and “rule” numbers, at this point you should only use the rule actions.

apply a fixed delay, send the frame to a Quality of Service ( QoS ) node, or send the frame to a connection oriented Address Mask, which in turn creates a new QoS node on the fly for new connections.

### Forward

The forward action immediately passes the frame to its destination without further processing of any kind. This action is used for traffic that is to be ignored or after all other rules have been exhausted. Other than a miniscule delay required for processing the rules, the frame is delivered as normal, as if the TSE was not present. Forwarding also automatically occurs when an invalid or undefined rule is processed. This would happen if an action branches to the “next rule” when the “next rule” was never configured. This behavior, however, *should never* be used as a substitute for an explicit “forward” action.

### Drop

Similarly, a rule can force the TSE to drop a frame. This should be used sparingly and with caution.<sup>12</sup> Dropping frames can cause severe disruptions to network traffic and network applications. Despite this admonition, the TSE can be used as an effective filtering mechanism. ACK filtering and similar operations can be performed by the TSE. The TSE can be used for filtering arbitrary frames, which is useful when trying to manage custom protocols that lack filtering support. The TSE should *not* be used to implement a firewall through filtering. Not only is this thoroughly ineffective, but the TSE is not designed to be a security product.<sup>13</sup>

### Fixed Delay

For extremely basic and usually ineffective bandwidth management, a rule can have the action of imposing a fixed delay on a frame. The delay is specified in 10µs units,

---

<sup>12</sup> You are *strongly advised* to use the filtering capabilities that come with NetWare and use the TSE for filtering as a last resort. See NetWare documentation on filtering for more information.

<sup>13</sup> Most modern firewalls are stateful, and are protocol aware. There is no way to use the TSE to implement a stateful firewall. The TSE works in conjunction with firewalling products, i.e. Novell Border Manager.

allowing up to a .65 second delay.<sup>14</sup> This is primarily used for testing the accuracy of the queuing mechanism, but it has other uses. This means of imposing a fixed delay bypasses the use of a QoS node. Consequently, no statistics are recorded by the TSE and the frame is directly submitted to the queue. Fixed delays are similar in behavior to fixed latency transmission media such as satellite connections. Characteristically, the round trip time for such systems is substantially larger than the latency associated with the time needed to encode the frame onto the transmission media.<sup>15</sup> A fixed delay per frame will reduce the effective data rate. But protocols like TCP and SPX, using the sliding window mechanism, overcome fixed latency. This severely limits the effectiveness of imposing fixed delays.

### QoS Nodes

The TSE's bandwidth management capabilities come from using QoS nodes to manage traffic. The QoS node encapsulates information about the desired data rate, threshold information, actual observed data rate, and TSE's housekeeping information.<sup>16</sup> The QoS node acts as a bucket into which traffic is placed, creating an individual, generally FIFO, queue for that traffic. Rules use QoS actions to direct the frame into a particular static QoS node. Several types of traffic can be managed as a group by using a number of rules to target *the same* QoS node. You could, for example, use a set of rules to lump all HTTP, SMTP, and AOL Instant Messenger traffic into the same group and apply a data rate to the group as a whole.

A number of static QoS nodes are configured using the Configuration Tool. New QoS nodes are created on the fly, initialized with information from a static QoS node that is used as a template. You assign the following:

---

<sup>14</sup> The integer value specified in the rule action is multiplied by 10 $\mu$ s to compute the actual delay. The value ranges from 0 to 65535 \* 10 $\mu$ s.

<sup>15</sup> The round trip time, RTT, for a satellite connection might be of the order of 500ms. Transmission speeds of 1 to 10 Mbps can be achieved on such links. A large IP frame might take only 1ms to be transmitted. The link transmits data at high speeds, but each ACK still requires 500ms.

<sup>16</sup> QoS nodes also hold the actual number of bytes and frames passed to it as well as the number of bytes or frames in the last threshold period. The ETA of the last frame is maintained by the TSE and is used in calculating the amount of delay to impose on the next frame to arrive.

- The number of bytes or frames per second
- The thresholds for turning the rate on and off
- Whether the threshold is a rate or an absolute value
- The sampling interval for assessing thresholds
- Whether the rate is to impose a fixed delay
- Whether the traffic can bank up unused bandwidth

When a frame arrives at a QoS node, the TSE updates the byte and frame counters. The actual rate of bytes or frames is then examined to see if the thresholds need to turn the rate on or off. If the data rate is not to be applied, the frame is forwarded. If the rate is to be applied, or threshold are disabled, the TSE initially computes a fixed delay value. This delay is based on the rate you specify and if that rate is by bytes or frames. For example, if you set a rate of 100,000 bytes per second, and the frame is 1000 bytes long, the delay will be  $1000 \div 100000 = .01$  seconds. The TSE thinks in microseconds. So a delay of 10000 $\mu$ s is computed. If the rate is a fixed delay rate, this delay is imposed on the frame. Normally fixed delay rates are *not* used for the reasons cited above.

■ **Using a by-bytes fixed delay rate will cause non FIFO queuing!** It would be possible for a smaller frame *received after* a larger one to exit the queue *before* the larger frame. This non-FIFO queuing is normally disruptive.

When using a *variable*<sup>17</sup> delay, the TSE takes into consideration the estimated time of arrival, or ETA, of the *last* frame processed for this QoS node. Suppose the last frame's delay should elapse in 3000 $\mu$ s, the final delay imposed on the current frame will be the ETA of the last frame + the current computed delay, 3000 $\mu$ s + 10000 $\mu$ s = 13000 $\mu$ s. The new delay becomes the new ETA value. Consider a sequence of frames sent very quickly, these would receive ever increasing cumulative delays. A pulse of closely spaced frames is "stretched out" in the queue to accommodate the desired data rate. This imposes a very accurate data rate on the frames feed to the QoS node. Unlike fixed delays, this *does* defeat sliding window protocols. The queuing in this case is always FIFO, so as to preserve the original reception sequence of the frames. Since all QoS

---

<sup>17</sup> We refer to the delay as "variable" because it varies depending on the timing of previously queued frames. The *data rate* is not variable, only the *delay* imposed on individual frames as they pass through the TSE. To impose a data rate of 100K per second, it might be necessary to apply a 5000 $\mu$ s delay to one frame and a 13000 $\mu$ s delay to the next, and so on.

nodes operate as independent queues, completely different data rates or rate strategies can be applied to selected traffic as desired.

### Address Masks

Suppose you want to limit the bandwidth available to *any* individual workstation. Rules can identify the general type of traffic desired. Then an address mask will separate the traffic from each workstation into its own stream of traffic with its own QoS node. This imposes a “policy” on any workstation’s usage. Similarly, you could limit the bandwidth available to any given TCP connection. You determine what a “connection” means by specifying what part of the frame is unique to a given connection.

Much like masks used in rules, address masks specify an offset type, offset value, and mask pattern. Each unique combination of the bytes selected by the mask spawns a new QoS node to manage all further traffic matching that signature. The new QoS node is initialized from a specified static QoS node. It is not possible to have various rates for various connections – this will be offered via NDS manageability.

The capability to set a policy on large number of connections or workstations is very useful. You can use a rule to identify all Internet traffic and apply a maximum data rate to each workstation. This would prevent any individual user from saturating your Internet connection. Using thresholds will allow users to have bursts of high-speed access without penalty, but have their throughput throttled back if they use too much. By using absolute threshold, download quotas on individual TCP connections can be imposed, defeating large downloads.

### Applications

Connection oriented limits can be used in the IP or IPX world to prevent a single user from overwhelming your NetWare servers. With the TSE you can perform sophisticated management of bandwidth, such as limit any individual workstation to 800K per second if they exceed 1000K per second for 10 seconds. The data rates can also be asymmetric, allowing unlimited data to be sent from the server, but apply limits to data received from the workstations. Use this to prevent slow servers from running out of cache or

screeching to a halt when a fast workstation writes a lot of data to the server.<sup>18</sup> *Very useful* when older servers clash with new workstations! Even that old '486 server can run the TSE to manage LAN bandwidth.

Another “interesting” application of the TSE is to manage broadcast traffic from workstations. While the TSE *cannot* prevent the workstation from transmitting the broadcasts, it *can* prevent the server from *seeing* them. This allows you to rate limit the number of broadcasts the server will see from any given workstation, preventing server wackiness if a workstation starts screaming broadcasts. Rate thresholds could activate a rate after a few seconds of screaming – allowing legitimate broadcasts generated at login to pass through the TSE, but throttle back unusually high broadcast rates. The same configuration protects IP based servers from seeing too many ARP, SLP, DHCP, etc traffic.

Another very useful capability is load balancing of SAP discovered servers. You could impose miniscule delays on the server's replies to Get Nearest... requests. This has the effect of pushing NOT-LOGGED-IN or “unlicensed” NDS connections to another server. After a period of time, the connections to the server will tend to be only those connections actually using resources on it. This is a great feature if you know you will be servicing a server... the server functions as normal, but defers these unnecessary connection to other servers, allowing the server to be taken down without superfluously impacting other user populations.

The same feature can be used to manually load balance SAP located services such as the NIAS and Border Manager IPIPX gateway. If you have three IPIPX gateways located by SAP, the “fastest” to respond will be found by the client. After a while you find that the number of connections to each gateway server is very skewed. Applying millisecond order delays to the SAP reply would allow you to make the “fastest” server a bit slower to respond, pushing new gateway clients to the other two servers. After a couple days of titration, the load can be spread very evenly. Other than the desirable effect of load balancing, there would be no real impact on server performance.

---

<sup>18</sup> The TSE cannot directly throttle the reception of data based on the amount of free cache memory. But the TSE can limit the total amount of data potentially written to disk. By setting a ceiling on the inbound bandwidth to match the servers ability to clear dirty cache, you can prevent the server from ever completely depleting cache.

Ever wonder how your network applications would work over a WAN? You can easily use the TSE to emulate a WAN environment by rate limiting all traffic through the LAN interfaces in the server. By setting the rate to that expected from the proposed WAN connection, your workstations would effectively be limited by the same bandwidth constraints. This works very well and can serve as an effective prototyping tool for justifying the cost of WAN links. You can actually show executive management the speed of the applications over, say a T1 link and then, a minute later, show them the speed of a 4Mbps service. Using rules to identify a small number of individual workstations, you can do the same in a production LAN environment. Only the specified workstation would have the rate imposed. This is also a must when trying to deploy NDS on a WAN, how long will it take NDS to sync or a backup to run?<sup>19</sup> Now you can find out.

## Future Development

Future development efforts include NDS manageability of the TSE, enhanced monitoring and logging, fault tolerance and redundancy.

The Beta TSE is designed to integrate with an NDS manageability add-on module. This module will associate network traffic with a given NDS user, group, OU, server, domain, host, or service. This supports bandwidth policies for these NDS entities, leveraging your existing investment in NDS. NDS information is commonly accessible from the TSEs running on different servers. TSE's will support clustering by sharing bandwidth statistics and quota information. This offers a way of centrally managing bandwidth through multiple routers as well as providing fault tolerance. The Beta TSE already has the internal capability to manage traffic on this level.

---

<sup>19</sup> Novell offers the NDS WAN Manager to provide better management of NDS traffic over a WAN. While the name of the product implies a traffic shaping capability, the WAN Manager only deals with NDS traffic. It does this by getting hooks into NDS' synchronization process, regulating the amount of bandwidth used to perform NDS synchronization and backup.

## Deployment Guide

The major steps in the deployment of the Beta TSE include identification of an appropriate test environment, installation of the software, configuration of the TSE, and finally loading and managing the TSE. The following sections describe the major steps of deployment in greater detail.

This is a ***beta release*** and that you may experience problems including server AbEnd. Even proper operation of the software may result in network disturbances, as the software's purpose is to limit the rate of network traffic. The software is provided to you free of charge, as-is, without warranty of any kind whatsoever. You take full responsibility for any damages of any kind that might result from the use of the software, this includes problems related to but not limited to improper operation of the software, program bugs, design defects, improper configuration, or incompatibility with your environment.

Should you experience problems, you are obliged to report these problems to the developer so that future versions of the software can be improved. This is **not** a promise to resolve any particular issues that may arise. Full details are provided in the License Agreement. Using the software binds you by the terms of the License Agreement.

## Planning

Ideally the Beta TSE will be installed in a completely isolated test LAN environment so there is zero risk to production systems and data. This is not always possible in practice. The scope of the TSE's interference with normal network activity *should* be limited to network traffic to and from the server running the TSE. You must evaluate the scope of a potential failure and weigh the benefits of using the Beta software against the potential for disruptions. Any NetWare server communicates with other servers for NDS purposes and NetWare servers can also act as a router, this must be taken into account when evaluating the scope of the risk. If an isolated test environment is unavailable, the TSE

should only be tested on servers which are frequently backed up and with a proper disaster recovery process in place. While it is unlikely that a failure of the TSE will result in data loss, this is an extreme, but potential risk with any Beta software.

## Server Requirements

	Minimum		Recommended	
	4.xx	5.x	4.xx	5.x
Processor	'386	'486	Pentium	Pentium
Memory ( MB )	32	64	64	128
Free Cache Buffers	2000	4000	4000	8000

The Beta TSE has been tested on servers as mangy as a '386/40 with 20MB of RAM and a generic ISA NIC. With that said, there are a few guidelines to follow to ensure your first experiences with the TSE are positive.<sup>20</sup> The server should have a '486 class CPU or better. The Beta TSE NLM uses about 1 MB of RAM. You need a minimum of 500 packet receive buffers, see below for details. You need to have about 2000 to 4000 free cache buffers available prior to loading the NLM.

- The Beta TSE only works with Ethernet network interfaces, it must *not* be used with FDDI, Token Ring, or ATM interfaces. The server may AbEnd if the TSE is bound to non-Ethernet interfaces.

During normal TSE operation, additional cache memory may be needed by the NetWare OS to create new Packet Receive Buffers to hold queued frames. Set the Minimum Packet Receive Buffers and Maximum Packet Receive Buffers parameter to accommodate the number of frames that *could* be queued at any given time. As a general rule, if the actual number of packet receive buffers used by the server

---

<sup>20</sup> These are recommended minimum values for servers dedicated to running the TSE. The specific applications you run may require more or less resources than those indicated above. The TSE itself does not require a lot of processing power nor a lot of memory. Additional memory is required by NetWare to improve performance and supply additional packet buffers consumed by the TSE's queuing.

increases, the high value should be set as the new minimum. The maximum should be double the minimum. Because the TSE will queue up to 8 seconds of traffic for any given connection, malicious attacks or screaming traffic can exhaust packet receive buffers.<sup>21</sup> Future version will provide limits to the number of en-queued frames and support an algorithm to accelerate de-queuing under these conditions. The depth of the queue will also be configurable.

### Interoperability

The TSE inter-operates with all major Novell products including Border Manager, NIAS, IP Gateway, GroupWise, Oracle database, and Novell Web servers. Other than the benefits of managing bandwidth, the TSE is completely transparent to these applications. The TSE implements a special low level interface to the OS called a “chained pre-scan protocol stack.” The TSE can inter-operate with other applications that implement chained pre-scan protocol stacks, such as compression stacks, provided that the compression stack is loaded and bound first. In other cases the TSE would normally be loaded and bound first, as when using the NOV\*IX gateway, which implements a pre-scan stack. You can determine if a pre-scan stack is in use by looking in MONITOR... Resource Utilization... Prescan Protocol Stacks and TX Prescan Protocol Stacks. The NLMs using these resources will be indicated. The TSE may conflict with very old *non-chained* protocol stacks.

### Installation

The program files are contained in three folders, SYSTEM, CLIENT, DOCS, which are part of a .ZIP file or CD-ROM distribution.

**IMPORTANT:** If updating the TSE with a newer version, you must unload the TSE prior to installation. You must back up all SHAPER.\* files from the SYS:SYSTEM directory on your server. Read the README

---

<sup>21</sup> Attacks such as the popular Trino and Tribe Flood Network attacks can cause this sort of problem. If the TSE is en-queuing excessive numbers of frames, you may be under attack. Having enough packet receive buffers will allow your server to ride through these attacks. The TSE is not a security product and is not a substitute for a firewall or managed firewall service.

files in the DOCS directory for information about the compatibility of the SHAPER.CFG file between versions of the TSE – they are generally NOT compatible. Copying a new version of SHAPER.CFG while the TSE is loaded may cause AbEnd.<sup>22</sup> Only copy the SHAPER.CFG file when the TSE is unloaded. The version of the .NLM and the version of the .CFG must match. You will have to recreate your configuration if the SHAPER.CFG file is incompatible between the existing and new version of the TSE.

1. Make backup copies of any replaced files.
2. Copy the contents of the SYSTEM folder to the SYS:STSTEM folder on the server. This folder contains the SHAPER.NLM, .SYM, and .CFG files as well as sample .NCF files.
3. The CLIENT folder contains SHAPER.EXE. This is the configuration tool. Copy this file to a convenient place on your management workstations.
4. Create a shortcut to SHAPER.EXE in the C:\WINDOWS\SendTo folder. This shortcut will allow you to right click on a configuration file and *Send it To* the configuration utility.
5. Copy the contents of the DOCS folder to a convenient location in your management workstation. DOCS, contains documentation in various formats.

## The Default Configuration

The default configuration of the TSE, as set by the distribution version of SHAPER.CFG, is to *pass all traffic*. Rule 0, QoS 0, and Address Mask 0 define this behavior. **Avoid editing these default entries.** Several sample QoS, Rules, and Address Masks *may* also be present in the higher numbered slots in the default configuration. You can modify or delete these samples. Details of the function of the samples, when present, are provided in the DOCS directory.

---

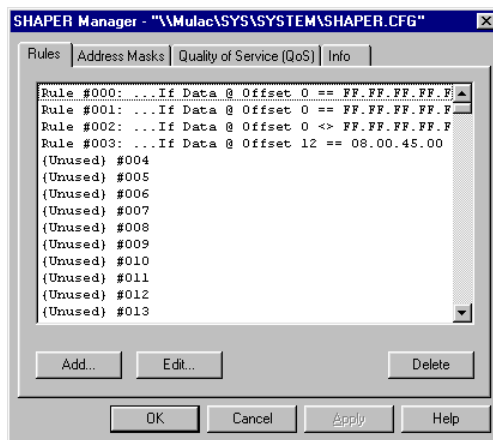
<sup>22</sup> Two issues cause the AbEnd condition. Copying the default configuration over top of your running configuration will delete items from the configuration, which can cause an AbEnd. If the versions of the configuration files are incompatible, the TSE may AbEnd.

## Using the Configuration Tool

The configuration tool edits the contents of the configuration file. The configuration file should *not* be manually edited, the file is in a non-human-readable, binary format.<sup>23</sup> Never use any means other than the Configuration Tool to edit the configuration file. When the TSE is running, changes made to SYS:SYSTEM\SHAPER.CFG are placed into effect as soon the file is changed. Deleting the file while the TSE is running will force the TSE write the *current running configuration* to a new SHAPER.CFG file..

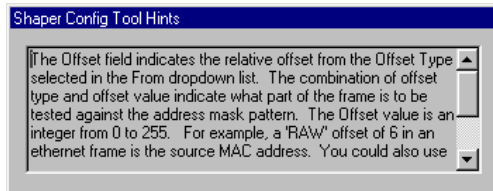
1. Start by making a **copy** of the SYS:SYSTEM\SHAPER.CFG file.
2. Find the **copy** in Windows Explorer and right click on it.
3. A context menu will appear. Choose **Send To** from the pop up menu.
4. The Send To menu should contain the Shaper Configuration Tool shortcut you created above.
5. Choose the configuration tool.
6. The configuration tool should start, revealing a rather strange splash screen.

## The Configuration Tool



The Configuration Tool has tabs for Rules, QoS nodes, and Address Masks. Each tab contains a list of available slots. An arguably cryptic text description identifies the item number and provides a semi-human-readable version of the item. The Info tab shows the Configuration Tool version and copyright information. *The configuration tool is still a work in progress, and admittedly is pretty horrible – this is being worked on.* Ultimately the TSE will be fully configurable via NDS and the configuration tool will only be used for debugging and testing.

<sup>23</sup> The configuration file is a memory image and should only be edited by the configuration tool.

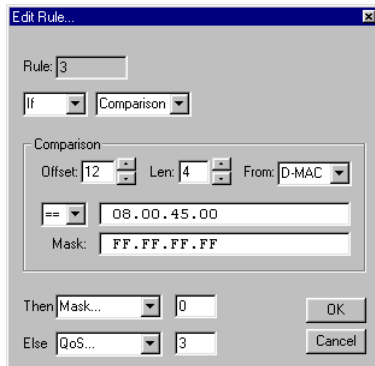


The smaller window shows the Hints panel. This window provides information about the currently selected item in configuration tool. When the Configuration tool has input focus, the Hints window will burn through any other window. *Eventually the Hints windows will be replaced by the standard Windows Tool Tips balloon help.*

The configuration tool does not have a standard “document style interface”... *yet*. There is no Save, Save As, Open, etc. menu options. To create a new configuration you must *manually copy an existing configuration file* and use the configuration tool to edit it. The configuration tool can be started more than once. This can be confusing since multiple Hints windows will open. *It is best that you edit a single configuration file at a time.*

All numbering in the configuration tool is zero based. You can add, edit or delete an item in the list by highlighting the item and using the corresponding button. You can also multi-select items to delete. Double clicking on an item, or an empty slot, will pop up an edit dialog. The details on using each edit dialog is described below

## Configuring Rules



The Rule Edit dialog modifies the contents of the selected rule. The purpose of a rule is to select and identify traffic based on its contents. This is accomplished by testing the frame's data for some desired piece of information. The items you might look for in a frame would include the source and destination MAC, IPX, or IP address, frame type, protocol ID, and protocol dependant information. With the IP protocol you might want to identify all TCP or UDP traffic, or specific TCP or UDP ports.

## Comparison

The “Comparison” section of the dialog specifies the offset type and value. Offset type and value specify where in the frame to look for your data. The data and mask fields indicate what to look for. The comparison operator indicates the type of comparison to perform. The comparison is evaluated for each frame processed by the rule. The outcome of the comparison determines which of the specified actions is taken.

## Data

The “Data” field contains a *dotted hexadecimal notation* value representing the data to look for. You can specify a pattern up to 16 bytes long. You can also test individual portions of the pattern and ignore others using masks, as mentioned below.

Dotted hexadecimal notation is a bit cumbersome when dealing with decimal numbers or strings, but is especially useful when you have packet traces from a protocol analyzer or from protocol reference guide. The calculator accessory in Windows, `calc.exe`, is a convenient tool for decimal to hexadecimal conversion. Dotted hexadecimal notation is a series of two digit hexadecimal numbers separated by periods, for example: `4A.75.64.79` which is the dotted hexadecimal notation for the ASCII string “Judy”. Any hexadecimal editor will make this job easy, for example UltraEdit 32. If looking for port 80, to identify HTTP traffic, you need to convert to hexadecimal notation, i.e.  $80_{10}$  is  $50_{\text{hex}}$  and so on. Be careful to perform the *proper* conversions.

## Mask

The “Mask” field is also a dotted hexadecimal notation field that indicates which bits in the Data field that are to be tested. When a Mask bit is 1, the corresponding Data bit is tested against the frame data. The Mask defaults to all FF’s, i.e. all bits set to 1, if left blank. This default Mask will make sure all portions of the Data are tested.

### **From ( Offset Type )**

The “From” drop down list selects the Offset Type. It allows the creation of rules that will work for various frame types and topologies. Just select the desired offset type from the drop down list. The “RAW” offset starts from the beginning of the raw frame data. Any raw offset will more than likely *not* work with multiple frame types or topologies. This is because the frame headers vary widely from frame type to frame type. The “S-MAC” and “D-MAC” offset types calculate offsets from the start of the source and destination MAC address. To look for a particular MAC address you need only select either S-MAC or D-MAC and use a 0 offset value. The “FRAME” offset type calculates offsets from after the first byte following the destination address. It is similar to the “RAW” offset type in that your comparison will be frame type dependent. The “FRAGS” offset type calculates offsets from the first byte of the protocol header. The “FRAGS” offset is immune to differences in frame type and is *the desired offset type to use* when looking at layer 3 data. The “ECB” offset type is a special purpose offset that allows comparisons to look at ECB data rather than frame data. Other offset types listed are unimplemented and are equivalent to a “RAW” offset.

### **Offset**

The “Offset” field holds the Offset Value which the TSE adds to the base offset specified by the Offset Type. To look at the 14<sup>th</sup> byte of the frame type independent portion of an IPX frame you would use an offset value of 14 and the FRAGS offset type.

### **Len**

The “Length” field is generated automatically as you modify the Data and Mask fields. You do not need to edit it, ignore it, leave it alone. Future versions will use this field to implement certain types of sub-string comparisons, which are as yet unimplemented.

### **Operator**

The “Operator” drop down list, shown in the figure above with “==” displayed, selects the type of comparison operation to perform. The list includes the standard mathematical style magnitude operators:  $>$ ,  $<$ ,  $=$ , and their negations,  $\leq$ ,  $\geq$ ,  $\neq$ . Normally the comparison will test for various byte patterns, like the signature of an 802.3 encapsulated

IPX frame, certain TCP port numbers, etc. However the magnitude operators:  $>$ ,  $<$ ,  $\leq$ , and  $\geq$  can also be used to compare integer values in frame data with the specified value. Most frame data is stored in “big-endian” format, where the most significant bytes come first. The TCP port number in an IP frame is an example of this format. Since the most significant bytes come first, the string style comparison will properly evaluate magnitudes. For example, you can test for TCP port numbers  $\geq 1024$  by specifying using the  $\geq$  operator and 04.00 pattern data with a FF.FF mask.

- • The meaning of the comparison is always “(frame data) (operator) (data + mask).” For example ( data at fragment offset 7 )  $\geq$  04 . 00 is representative of the intent of the comparisons used in the TSE.<sup>24</sup>

### Caveats

There are some limitations and caveats to comparisons. First, the magnitude of little-endian numbers cannot be easily compared. The ability to specify big-endian or little-endian format is a future enhancement to be made to the TSE. The sub-string operations, \$, &, !\$, !& are not implemented in this Beta version of the TSE, do not specify these operators as their behavior is undefined.

There is a difference between incoming and outgoing frames concerning the work required to perform comparisons. Incoming frame data is in raw off-the-wire format, and so is easily tested. Outbound frames, as seen by the TSE, have yet to be fully constructed by the NetWare OS. The TSE must “fill in the blanks” to make the outbound frame look like a raw inbound frame so that your comparisons work the same on both. Using the FRAGS offset type avoids the overhead of constructing the missing frame data. Other offset types will result in increased overhead in processing rules. The additional overhead for outgoing frames does not normally translate to outrageous CPU utilization except when a ridiculously high number of rules are processed for large numbers of frames. If you encounter performance issues on outgoing frames, it is normally possible

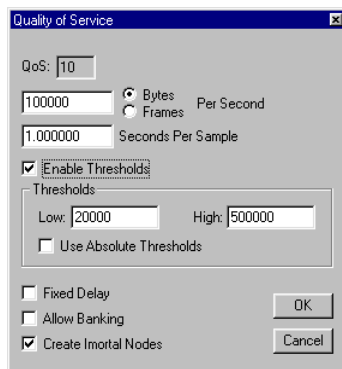
---

<sup>24</sup> The comparison concept for the TSE was shamelessly stolen from the Nortel Networks SpeedView Centillion Switch Configuration Utility. The switch offers filtering capabilities manageable through SpeedView. Unfortunately they never really tell you if your comparisons put the frame data as the l-value or as the r-value. For the TSE, the frame data is the l-value, and the user-supplied data and mask make up the r-value for binary comparison operations.

to manually rearrange the sequence of rules executed, placing the most limiting rules first. This is a good practice for any set of rules.

- The “If” and “Comparison” dropdown boxes must be left in the default state. Future versions of the configuration tool and TSE will use these. For the moment, just ignore these.

## Configuring QoS



Use the Quality of Service Edit dialog to define the data rates which Rules and Address Masks refer to. Each QoS entry or node acts as an individual queue or bucket into which traffic is dumped. Several rules can divert traffic to a given QoS node, allowing different types of traffic to be managed as a single data stream. Address Masks use QoS nodes in a different way. The QoS node acts as a template for the creation of new QoS nodes, one for each individual connection identified by the Address Mask. This is how individual connections can be assigned a data rate, creating a separate queue for each connection.

### Data Rate / Sample Period

The QoS entry provides the information needed by the TSE to apply a data rate. Rates can be specified in terms of bytes or frames per second. The actual data rate is computed by the TSE on a periodic basis. The frequency with which the actual data rate is computed is set by the Sample rate, specified in seconds. The TSE evaluates thresholds, if any, each sample interval. The sample interval, therefore, controls the *sensitivity* of thresholds to rapid changes in actual bandwidth usage, modifying threshold reaction time.

## Thresholds

The Low Threshold tells the TSE to turn off the data rate if the actual bandwidth used falls below the specified rate. When the rate is “off” all traffic passes unhindered through the TSE – as though the TSE is not present. When the actual bandwidth used exceeds the High Threshold, the rate is turned on, and the data rate specified is enforced. Absolute thresholds do not monitor the *rate* of traffic, but rather the *total amount* of traffic since the QoS node was created. Absolute thresholds are used to apply the data rate after a certain total volume of traffic has been exceeded. Since the absolute counts are always increasing, once the rate is turned “on” by an absolute threshold, it never turns off and persists the life of the QoS node.

## Fixed Delay Option

The Fixed Delay checkbox *is not normally used*, and it can cause frames to be queued out of sequence when used with a byte rate. Checking this box tells the TSE to calculate a delay based on the indicated rate, but *not* consider the ETA of the last frame. For example, setting a rate of 100 frames per second in conjunction with Fixed Delay would assign a 10ms delay for each frame. Using this option with a byte rate will produce different delays based on frame size. It is possible for a smaller frame received *after* a larger one to be de-queued first. This disrupts the original reception order of the frames and is normally undesirable and disruptive. The primary use of this option is to impose fixed latencies. Fix latency can also be applied directly by the Delay action in a Rule. The advantage of using a QoS node rather than a Delay action is that the TSE gathers statistics about the traffic. This option is useful in several special applications discussed later.

## Allow Banking Option

The Allow Banking checkbox tells the TSE to allow the use of the “unspent” portion of the data rate. Suppose a data rate of 100K per second is specified, but only 50K per second is actually used – the unused portion of the rate can be banked up against future burst usage later on. This applies a “fair” data rate, from a user’s perspective, but allows large peaks in bandwidth utilization when the user taps into their banked bandwidth.

For example, if the above connection were running for 20 minutes, over 60MB of bandwidth would be in the bank. The user could conceivably generate any arbitrarily high bandwidth utilization until the 60MB is exhausted. Bandwidth banking can be used when small data rates are applied in an effort to impose bandwidth quotas.

If this option *is* used, it is advisable to have some other governor on bandwidth, such as managing both interfaces in a routing situation separately. One interface could manage bandwidth by connection, and the other as an aggregate, allowing an overall rate to be applied that would throttle an individual user with a lot of banked bandwidth. Another way to defeat or limit the potential bursts from banking is using an absolute TTL in the address mask. This expires the QoS node after a period of time – forgetting any banked bandwidth. In any event the total amount of banked bandwidth cannot exceed 60 minutes of the rate specified.<sup>25</sup> Currently there is no direct way to assign an arbitrary ceiling on the amount of banked bandwidth per connection or QoS node. This TSE wide tunable parameter applies to all QoS nodes, no matter how they were created.

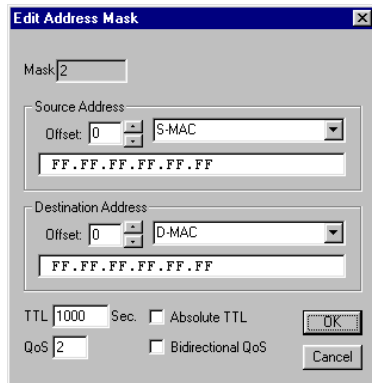
### Create Immortal Nodes Option

The Create Immortal Nodes checkbox tells the TSE to create QoS nodes that live forever. QoS nodes created by Address Masks will live only as long as the TTL on the address mask specifies. Immortal nodes and any associated connection state data will not be removed by periodic housekeeping functions. The same effect is achieved by setting the TTL on an address mask to be ridiculously high.

---

<sup>25</sup> The 60 minute horizon for bandwidth banking is based on a technical limitation of the TSE. The TSE thinks in microseconds and uses a 32 bit number to remember the last time a frame was sent. A positive ETA value means there are frames currently queued for that QoS node, there is no banked bandwidth. A negative ETA value indicates that no frame has been sent during that period. The amount of banked bandwidth is equal to the negative ETA **J** data rate. The maximum negative ETA is about 4000 seconds. 3600 seconds was selected as the upper limit. Future versions will allow this to be configured.

## Configuring Address Masks



The Address Mask Edit dialog allows you to configure the address mask information used to manage connection oriented traffic. The address mask specifies the portions of a frame which uniquely identify that frame as belonging to a certain connection. A “connection” can mean whatever you want it to. Specifying the source and destination MAC address, a connection would be all data passing from one host to another. Specifying source and destination IP address and TCP port would assign the same data rate to each TCP connection individually. Any arbitrary frame contents can be specified for the source or

destination “address.” This is a very flexible means of further separating a group of traffic identified with rules into individually managed data streams. Offset types, offset values, and masks in the Address Mask edit dialog work in a similar fashion to those in the Rule editing dialog.

### Offset / Offset Type

The Offset and offset type as well as the mask specified for the source and destination address work the same way they do for rules. They identify the part of the frame you want to deal with. See the corresponding descriptions of these fields above for more information. The masks can be up to 16 bytes long, for a total of up to 32 bytes. A mask can be created to, for example, mask off the IP address and TCP port number. The source and destination masks do not have to be the same pattern. For example, the “source” address could be used to identify the IP traffic type, IP, TCP, ICMP, etc. and the “destination” can be used to mask off the source and destination IP address. The sum of both masks produces a unique “key” used in managing the connection and identifying further frames for that connection. The terms “source” and “destination” are, in fact, meaningless – either mask can be used in whatever way you want.

## **TTL ( Time To Live )**

The TTL field indicates the Time To Live for a newly created connection. If no frame for a particular connection is received in the TTL period, the connection and its QoS information are forgotten. This has no real impact on the connection between the hosts involved, only in the TSE's housekeeping of connection information. If no traffic for a particular connection is flowing, it is pointless for the TSE to track the connection's bandwidth usage. After the TSE "forgets" a connection, the next frame to arrive for that particular set of source and destination address will create a new connection entry and QoS node to manage it. Most TCP connections would have relatively short TTLs, up to 2 minutes. For managing host to host traffic, the TTLs might be longer to prevent the TSE from frequently forgetting and reacquiring the connection.

Overall, it is desirable for the TSE to keep track of as few connections as possible. The fewer connections, the less work the TSE expends in analyzing incoming and outgoing frames. It takes about 4 times as long to learn a new connection as to locate an existing one. The goal of lowering the TTL value is to reduce the number of connections while not forcing the TSE to frequently relearn recently deleted entries. For typical TCP connections related to WWW traffic, the lowering of the TTL will have a positive effect because the connections are ephemeral.

## **Absolute TTL Option**

The Absolute TTL checkbox tells the TSE to expire the connection after the TTL expires, regardless of the presence of traffic for the connection. Setting an absolute TTL of 20 seconds will make the TSE forget the connection after 20 seconds, even if the connection has recent activity. This is useful in conjunction with QoS Absolute Thresholds. As mentioned earlier, when a QoS node has an absolute threshold, the rate will be activated by the crossing of the High Threshold, but will *never turn off*. Setting an absolute TTL causes any QoS nodes with their rate in the "on" state to be removed periodically, allowing a new allotment of bandwidth to be yielded to the workstation.

### QoS

The QoS field specifies the number of the QoS node, in the QoS Tab of the configuration tool, to use as a template when creating new QoS nodes for individual connections. The QoS node specified is the template used to dynamically created QoS node.

### Bi-directional QoS Option

The Bi-directional checkbox tells the TSE if it should manage each direction of the connection's conversation individually or collectively. Checking this box indicates that a single QoS node should manage all traffic for the connection. This causes the transposition of the source and destination masks on outbound traffic resulting in the same "key" value. When unchecked, each side of the conversation is managed individually. The basic difference is that bi-directional mode of operation makes the data and acknowledgements compete for the same bandwidth, more closely matching the performance of a modem like connection. When each direction is managed separately, it would seem that asymmetric data rates could be applied – though the TSE is capable of this, the configuration utility does not provide a means of managing this feature.

### Operation

- To make management easier, the Beta TSE should be bound to interfaces which were configured with logical board names. Use of INETCFG will automatically create named logical boards and is strongly recommended.

Once a configuration has been built, you are ready to use the TSE on your server. After loading the NLM from the server console you use additional console commands to bind the TSE to the desired logical boards. Console commands also select the first rule to be executed by the TSE for each bind. Normally these commands can be reduced to a .NCF file that loads the NLM and completes the runtime configuration. A common startup sequence might look like:

## ShapeShifter Beta TSE 1 -Network Manager's Guide

```
#
# Load and bind the TSE to desired logical boards
#
LOAD SHAPER
BIND SHAPER TO E100B_1_EII
BIND SHAPER TO IBMFE_1_EII
#
# Tell the TSE which rules to execute first for each half-bind
#
SHAPER BIND 0 RULE 5
SHAPER BIND 1 RULE 20
SHAPER BIND 2 RULE 15
SHAPER BIND 3 RULE 25
#
# Tell the TSE to place the binds in an "UP" state
#
SHAPER BIND 0 UP
SHAPER BIND 1 UP
SHAPER BIND 2 UP
SHAPER BIND 3 UP
```

The above sequence loads and binds the TSE to the Ethernet\_II logical board on both the IBMFE and E100B adapters in the server. Each bind command internally creates two separate binding entries for the TSE. Even numbered binds are the *reception* side of the binding and odd numbered binds are the *transmission* side of the bind. Each of these “half-binds” can have different sets of rules defined. The Beta TSE can be bound to a maximum of 4 logical boards. This limitation is a Beta restriction. In our case the binds look like:

<b>TSE Bind Number</b>	<b>Logical Board</b>	<b>Direction</b>
0	E100B_1_EII	Receive
1	E100B_1_EII	Send
2	IBMFE_1_EII	Receive
3	IBMFE_1_EII	Send

This same information can be obtained from the server console using the SHAPER DEBUG BINDS command. This command displays each TSE bind number, the logical board, and the traffic direction, as well as various statistics.

Once bound to a particular logical board, the bind is in the “bypass” state. In this state the TSE will forward any traffic without examining it. The bind can also be in the “down” state where all traffic is dropped. This will cause all traffic for the specific bind to be discarded. A third bind state is the “up” state, where the TSE manages the traffic. The SHAPER BIND nn UP | DOWN | BYPASS console command modifies the state of the specified bind. All binds can be placed in the “bypass” state by using the SHAPER BYPASS console command.



## Sample Applications

The following section outlines several sample applications for the Beta TSE. Each is to be taken as a separate stand-alone configuration – it is assumed that the TSE does nothing else. In more complicated configurations, you integrate several sets of rules and tie them together so that the desired effect is achieved. Use these as seeds for dreaming up your own applications and as templates for your initial deployment of the TSE.

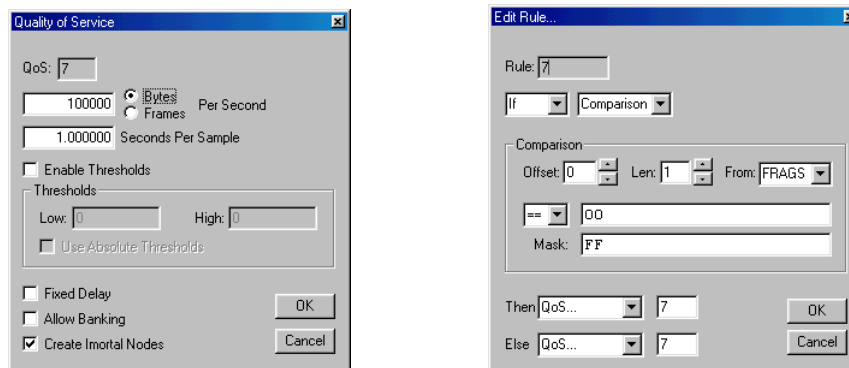
Until you become familiar with the operation of the TSE and the Configuration Tool, you should use the following steps when performing any configuration changes:

1. To be safe, work *without* the TSE loaded. Unload it if it is running.
2. Copy the SYS:SYSTEM\SHAPER.CFG file to another location to be edited.
3. Copy the file again to make a *backup* copy of the .CFG.
4. Right click on the *copy* of the .CFG a popup menu will appear
5. From the popup menu, choose Send To
6. Select the Configuration Tool
7. { Make your changes in the Configuration Tool }
8. Copy the edited .CFG file over top of SYS:SYSTEM\SHAPER.CFG
9. LOAD the SHAPER.NLM from the console
10. { Manage the binding to activate the new configuration }

If the TSE is loaded, any changes to the configuration file are automatically loaded and activated. This can be undesirable – especially when deleting items from the configuration. Unloading, changing the configuration, and reloading the TSE is the safest method for beginners.

## Case 1 – Rate Limit An Interface, Logical Board

It is often desirable to limit the total bandwidth flowing through a given interface. Unless the network infrastructure supports this, it is often impossible to do so. The TSE can easily be configured to rate limit the bandwidth of a physical interface or a logical board. First, you need to define a rule to *identify* the desire traffic. In this case *all* traffic for a specific logical board is to be managed. Create a rule that performs a trivial test, with the Else... and Then... actions directing the traffic to the QoS Node that applies the desired rate. A suitable rule would tests if the first byte from the Fragment offset is 0.



Pick an empty QoS Node and set the desired data rate. The dialog above assigns a 100K per second data rate. You can enable thresholds, bandwidth banking and other rate limiting features as desired. Once a QoS Node has been created, configure the rules that will direct traffic to the QoS Node.

Pick an empty Rule entry from the Configuration Tool and direct the rule to send the traffic to the QoS Node. In the example shown above, we look for 00 in the first byte of the fragment. It really does not matter what we test for so long as both actions direct the traffic to the QoS Node. In our case this will be QoS Node 7.

After saving the configuration we bind the TSE to the interface we want to manage and tell the TSE to start with our new rule. In our case the rule we just created is Rule 7. From the server console we would use the following commands:

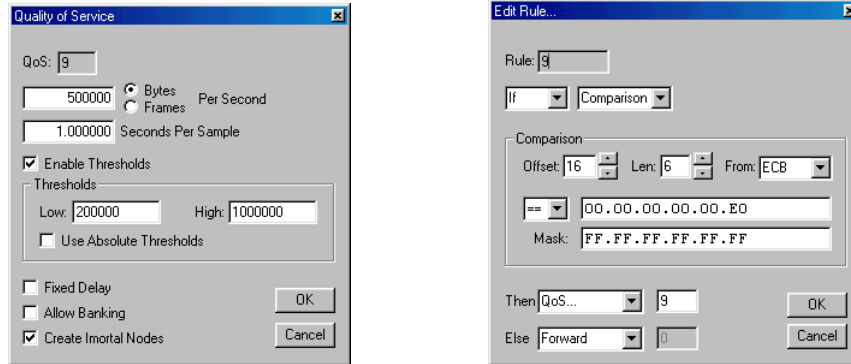
```
LOAD SHAPER
BIND SHAPER TO IBMFE_1_EII
SHAPER BIND 0 RULE 7
SHAPER BIND 1 RULE 7
SHAPER BIND 0 UP
SHAPER BIND 1 UP
```

Now all the ETHERNET\_II frames through our first IBMFE interface will be assigned a 100K per second data rate. Place these commands in an NCF file for easy use. This example can be easily retooled to manage all traffic through an interface by binding the TSE to all logical boards for a given interface.

### **Case 2 – Rate Limit A Protocol**

It is often desirable to limit the total bandwidth used by a particular network protocol.

Taking Case 1 and tweaking it you can identify and apply a data rate to any arbitrary protocol. The concept is exactly the same except we use the rule to look for IPX over Ethernet 802.2 frames. The question then is, “How do I test for an 802.2 frame with an IPX frame in it?” The Event Control Block, or ECB, includes a Protocol ID field at offset 10h, which is 16 in decimal. The Protocol ID is a 6 byte field, padded with leading zeros. E0h is the protocol ID for IPX over Ethernet 802.2 frames. So we look for 00.00.00.00.00.E0 at offset 16 in the ECB.



We define a slightly more complicated data rate with QoS Node 9. Thresholds throttle back the available bandwidth after it reaches a peak value. When more than 1000K per second is sustained for more than a second the 500K per second rate is enforced. After the volume of traffic falls below 200 K per second the rate is turned off.

The rule must test the entire contents of the protocol ID field including the zero padding. The rule sends all matching frames to QoS Node 9. All other frames are forwarded. Obviously we could test for other types of frames by creating more rules and using the Else... conditions to string them together in a sort of if...then... bucket brigade.

To activate this configuration, use similar commands as used in Case 1. Just bind the TSE to the desired Ethernet 802.2 logical boards.

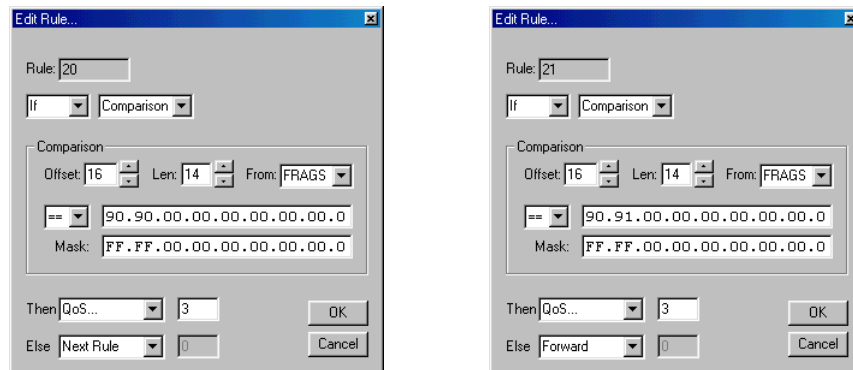
## Case 3 – Rate Limit IPX IP Gateway Traffic

The above cases show how to identify and rate limit all frames for a given protocol. By applying a couple additional rules, the TSE can identify the IPX side of the IPX IP Gateway traffic. This traffic can be rate limited, effectively regulating the total IP traffic passing through the Gateway.

First we need a rule to identify the IPX traffic in general, such as that constructed above. Then we need another couple rules to identify the IPX IP Gateway traffic. The nice thing about IPX IP Gateway traffic is that the IPX side uses IPX socket 9090h and 9091h only. This makes our job real easy. The 9090h traffic is used for encapsulated UDP traffic, while 9091h is used for TCP. So we really need a rule that identifies IPX frames

where the source and destination socket is either 9090h or 9091h. These two socket numbers vary only by the last bit, so we use a mask to ignore the last bit. Here are the rules we would need in either case.

Without using masks to ignore the last bit we need two rules, one for 9090h and one for 9091h. Here they are:

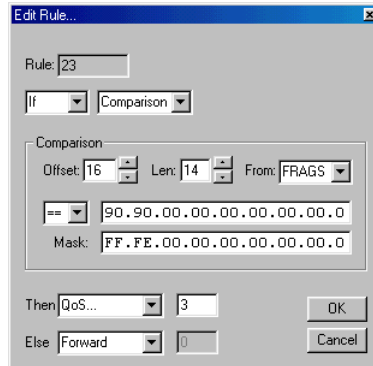


The data and mask fields ( that are obscured in the figures ) look like:

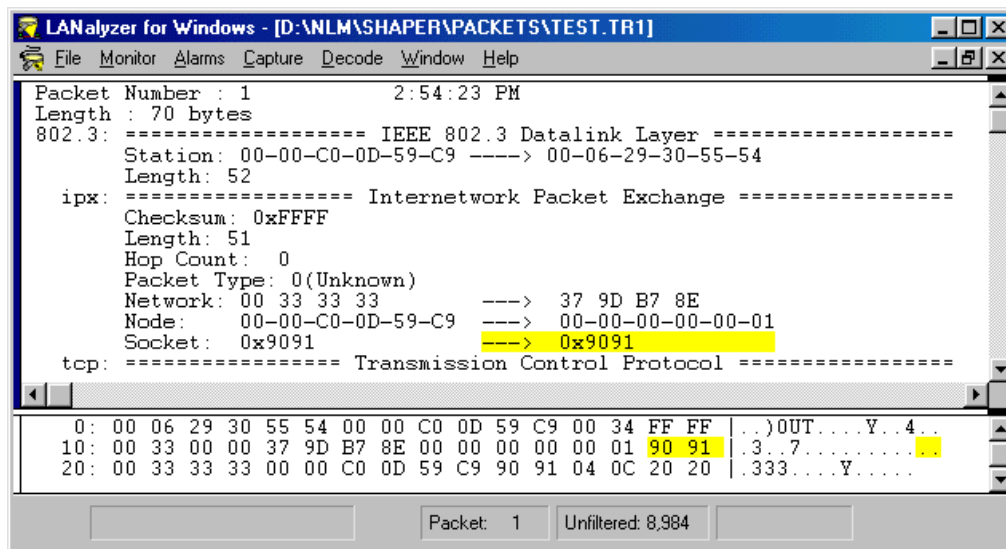
```
Data: 90.90.00.00.00.00.00.00.00.00.00.00.90.90
Mask: FF.FF.00.00.00.00.00.00.00.00.00.00.FF.FF
```

```
Data: 90.91.00.00.00.00.00.00.00.00.00.00.90.91
Mask: FF.FF.00.00.00.00.00.00.00.00.00.00.FF.FF
```

We reduce this to a single rule by masking off the socket values with FF.FE rather than FF.FF. The FF.FE value masks off all but the last bit, so the rule matches either 90.90 or 90.91 which differ only in the last bit. Here is the rule:



But how do we know what to mask off anyway? Well, one way to figure this out is to use a protocol reference manual. A good protocol analyzer is also invaluable. Using a packet trace is the best way because you can be sure exactly what types of frames are present in your environment and what they look like. Here is a LANalyzer dump for an IPX IP Gateway frame:



In the lower portion of the window we see the hex dump of the frame. The highlighted portion corresponds to the destination IPX socket number. Note that this is 16 bytes in

from the start of the IPX header. So we use a fragment offset value of 16. By using a fragment offset, these rules will work for any frame type.

Unfortunately the encapsulated IP frame that follows the IPX header is somewhat useless if we want to further rate limit, for example, just WWW traffic through the gateway. The best way to do that is to rate limit IP frames destined for the server itself.

## Case 4 – Rate Limit HTTP Traffic

If you have a Novell based web server, such as the Netscape Enterprise Server, it would be nice to limit the total amount of traffic the server can generate. This is actually very easy. The protocol ID for Ethernet encapsulated IP frames is 00.00.00.00.08.00. So a rule is needed to test the ECB for this protocol ID as we did in Case 2. Then you need to test these IP frames for traffic leaving the server over port 80.

If you are in an IPv 4 environment, i.e. are using standard IP, we can assume that all IP frames seen by the server have the standard IP headers. If this is not the case you will have to test to see if the frame is IPv4. This is easy, as the first byte of the IP header, at fragment offset 0, is 45h. If the ECB indicates an IP Protocol ID, you'll certainly see the 45h signature. So it is not really worth testing for it. Now that we have convinced ourselves that we do not have to test the IP header for version, we can reduce the rest of the criteria to a 16 byte span that can be tested with a single rule. Here is how we do it.

From the beginning of the IP header, we need to test for the 06h signature for TCP at offset 9. The source IP address starts at offset 12. The source port is at offset 20, for HTTP this is 0050h. Using a fragment offset of 9, the data and mask would look something like this:

```
Data: 06.00.00.AA.BB.CC.DD.00.00.00.00.00.50
Mask: FF.00.00.FF.FF.FF.FF.00.00.00.00.FF.FF
```

The underlined section of the data field you will fill in with the hexadecimal notation for the IP address of the web server. This is a very flexible template that can be used to identify a variety of IP traffic. Modifying the mask portion corresponding to the source

IP we can identify traffic by subnet. By tweaking the source port we can identify many different services.

So is the server's IP address is 192.168.0.11, we would convert it to C0.A8.00.0B. Integrating this into the Data and Mask we arrive at:

```
Data: 06.00.00.C0.A8.00.0B.00.00.00.00.00.50
Mask: ff.00.00.ff.ff.ff.ff.00.00.00.00.ff.ff
```

So using two rules, one to identify IP frames using the ECB's Protocol ID field and a second rule to identify the specific traffic we want, we can control very specific types of traffic based on transport type, address, and port.

### Case 5 – Rate Limit A Traffic To / From A Given Domain

Sometimes it would be nice to limit the total throughput to a given range of addresses or an entire IP domain or subnet. This is very similar to Case 4, only with much less restrictive criteria. Suppose we want to limit the bandwidth to or from 192.168.12.0/24. We want UDP and TCP frames to be included. We will need two rules, one to identify any traffic sent from the network, and another to identify any traffic sent to it.

From the beginning of the IP header, the source IP address starts at offset 12, followed by the destination address at offset 16. The data and mask, both with respect to a fragment offset of 12 would look like:

```
Data: AA.BB.CC.DD.00.00.00.00
Mask: NA.NB.NC.ND.00.00.00.00

Data: 00.00.00.00.AA.BB.CC.DD
Mask: 00.00.00.00.NA.NB.NC.ND
```

The underlined section of the data field you will fill in with the hexadecimal notation for the IP address of the network and the network mask. So if the network is

192.168.12.0/24, convert it to **C0.A8.0C.00** and a network mask of **FF.FF.FF.00**. Integrating this into the template above we arrive at:

Data: **C0.A8.0C.00.00.00.00.00**  
Mask: **FF.FF.FF.00.00.00.00.00**

Data: 00.00.00.00.**C0.A8.0C.00**  
Mask: 00.00.00.00.**FF.FF.FF.00**

So we need only three rules to accomplish the task. The first rule identifies IP frames based on the ECB's Protocol ID field. The other two look for IP frames with a source or destination address in the network we want to rate limit. A similar process can be used to manage traffic based on port or based on a combination of port and IP address.

In the example above we use the same offset for both rules. This is obviously not necessary and was only done so that the relative position of the source and destination addresses could be seen in the diagrams. The leading 00.'s in the second rule could be eliminated if the offset value were set to 16 to compensate. The amount of time required to process the leading 00's in miniscule.

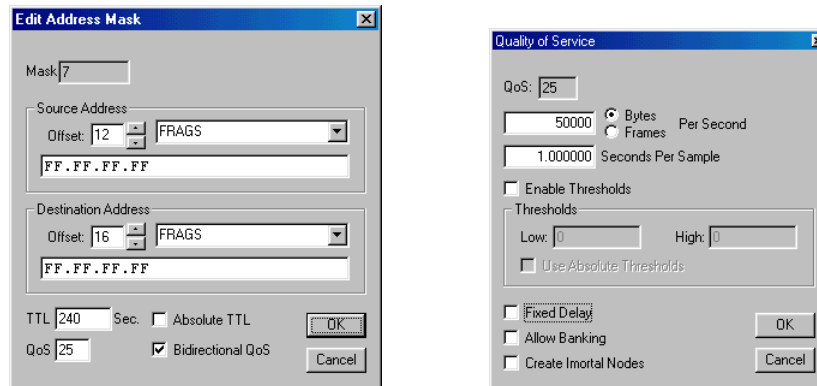
### Case 6 – Connection Oriented Limits: IP Traffic

It is often desirable to impose broad policies on the maximum amount of bandwidth available to a given client. We already know how to identify different types of traffic and apply a data rate to them. With the help of address masks we can go one step further to apply rates to individual workstation or connections, thereby imposing bandwidth policies.

In this example we want to rate limit the IP throughput for any individual workstation to some specific value. The key to using connection-oriented rates is to determine what you mean by a "connection." In our case we will take a connection to mean "a data stream consisting of all frames between any two arbitrarily chosen IP addresses." In the context of managing bandwidth to a given server, this will limit the throughput available to any

given workstation. In the context of a router, the rate is applied to individual conversations between any two hosts that pass through the router.

As in previous examples, you need to identify the broad category of traffic you want to manage, so we will use a rule to identify IP traffic based on the Protocol ID field in the ECB. The rule uses a Mask action to send the traffic to an Address Mask. We construct an address mask that masks off the source and destination “address” used to determine which connection a frame belongs to. In our case the source and destination addresses are the source and destination IP addresses. From previous examples we know the source address is at fragment offset 12 and the destination at fragment offset 16, and both are 4 bytes long. So you define an address mask to indicate this. Here is an example:



The address mask selects the source and destination IP address using similar fields to the Rule dialog we have used in previous examples. The TTL field selects the Time To Live, which tells the TSE how long to keep tabs on an inactive connection. In this case the TTL is set to 4 minutes. If no new frames are received for a given connection in that period, the TSE forgets about the connection. This has not real effect on the underlying connection, only on the TSE’s housekeeping. The TSE operates more efficiently if it has fewer connections to monitor.

The Bi-directional QoS checkbox tells the TSE to manage both sides of the conversation with a single QoS node. This has the effect of making inbound and outbound data compete for bandwidth. We also specify the QoS node to use as a template for new connections. The QoS Node shown above is pretty standard with *one* exception. The Create Immortal Nodes checkbox is unchecked. This allows the TSE to forget the QoS

node when the TTL expires. Without doing this, the TTL in the Address Mask is ignored.

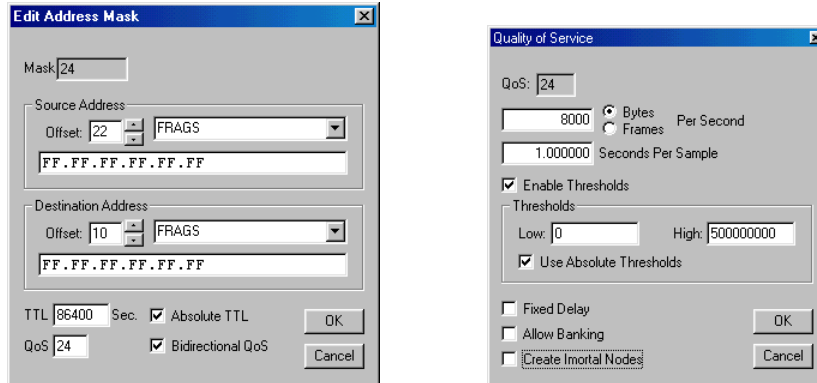
The end result is that the effective IP data rate between any two workstations is limited to 50K per second. The effect is to limit the amount of traffic from the server to the workstations. If the server acts as a router the effect will be to limit the traffic between workstations on opposite sides of the router. Obviously the workstation can be talking to several hosts, each with a different IP, allowing much greater bandwidth to arrive at the workstation than the 50K a second allowed for an individual connection.

### **Case 7 – Connection Oriented Limits: Quotas**

The TSE makes it easy to implement bandwidth quotas. Suppose we want to provide each workstation with 500 MB per day of bandwidth? The question becomes “what happens after they run out?” The QoS Node supports an elastic form of quotas by using absolute thresholds. After the threshold is reached, the QoS node enforces the specified data rate. So a user would be able to download up to 500 MB without restriction, but once this is exhausted, a very low data rate could be applied. A data rate of 8 K per second is sufficiently low. Once enabled, the data rate would never switch off. So a workstation would have to be turned off for a period of time to let the connection time out. There are some obvious limitations to using *only* the QoS to implement a quota.

To enforce an hourly or daily quota, you can enable an absolute TTL via the Address Mask. An absolute TTL will clear out the connection ( and the associated QoS Node ) after the time specified. Setting this to 24 hours would have the effect of resetting the QoS Nodes for all connections every day, giving the connection a new pile of bandwidth to burn.

Using the rules in Case 3 to identify the IPX side of the Gateway traffic, and adding a QoS Node and Address Mask this can be used to apply quotas to IPXIP Gateway users. A similar method can be used for any connection oriented rate. Here is what it would look like:



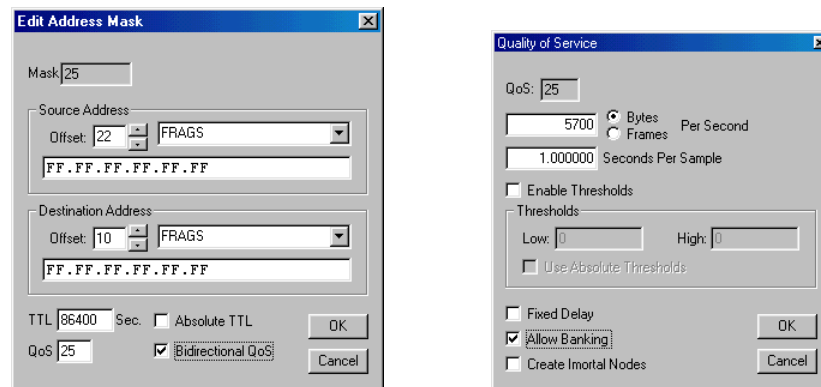
Note that the Create Immortal Nodes checkbox is *not* checked. If this were checked, the connections would never clear. Since the rules in Case 3 supply us with IPX frames, the offsets in the address mask are the source and destination node address in the IPX header. The node address of the server is 00.00.00.00.00.01, but the only server traffic we will see would be the server running the TSE. But this is a situation that might not be true for any particular application, such as routing. The Bi-directional QoS checkbox is checked to tell the TSE to manage inbound and outbound traffic as a single group. This makes uploads and downloads eat away at the 500 MB per day quota. Once the quota is exhausted, a limit of 8 K per second is imposed.

At 8 K per second, it is possible to move about 680 MB per day! So to impose a real quota, it would be necessary to cut down the effective rate to about 2 to 4 K per second and decrease the threshold so that the total adds up to the quota desired. Using a 150 MB threshold and a 4 K per second rate a quota of 500 MB per day can be enforced. Such high quotas would hardly be noticed by the average user, but would be a rude awakening for bandwidth hogs. An extremely low data rate would force the TSE to drop frames and would interfere with the excessive traffic – resulting is a hard quota.

## Case 8 – Connection Oriented Limits: More Quotas

A 500 MB per day quota could also be enforced by a blanket 5.7 K per second data rate with bandwidth banking. The beauty of bandwidth banking is that it favors burst traffic like web surfing, while effectively clamping down on constant traffic like audio and

video feeds and mass downloads. Every byte of unused bandwidth can be instantly used to perform high speed downloads. That 5.7 K per second limit adds up to a lot of bandwidth, over 20 MB per hour. So after an hour of idle time, the user could download a 20 MB file at full speed. The TSE allows up to an hour of bandwidth to accrue, so you could not bank up more than 20 MB at 5.7 K per second. User downloading a batch of MP3 files will quickly exhaust any banked bandwidth and then face the indignity of downloading stuff at 5.7 K per second all day long. While this does not prevent the download of these files, it does discourage the activity by making it painfully slow. A QoS Node which you would use to accomplish this might look like:



We use a similar address mask as in the previous case, except the TTL is not an absolute TTL. After a day of inactivity the connection entry and QoS node will be dropped. This can be set to any desired value, though a TTL of less than 1 hour would defeat the bandwidth banking scheme.

If you did *not* check the Bi-directional checkbox, two separate QoS Nodes would be used to manage each side of the connection. This would provide a separate bandwidth bank for uploads and downloads, which might be useful for users who occasionally upload large files. You could also apply a different rate scheme if inbound and outbound frames.

## Case 9 – Combining Techniques

Sometimes it is not possible to achieve the desired result with just a single rate limiting scheme. There are some tricks that can be used in the case of boundary routers, where all traffic entering one interface leaves another, or in the case of Gateways where inbound traffic leaves as outbound traffic. In these cases you can employ one rate limiting scheme on inbound traffic and another on outbound traffic to achieve a combined effect of the two.

For example, you could rate limit individual workstations on the IPX side of an IPXIP Gateway, while using another technique to rate limit all IP traffic on the IP side of the Gateway. This would allow you to do things like limit the total IP throughput to a given Gateway to be 200 K per second, but impose a 1 GB per day download quota on any individual user. Similarly you could identify all “normal” IP traffic ( HTTP, SMTP, POP, etc. ) and let it pass unhindered, but rate limit the rest to 50 K per second. This combination of two different types of rates is very powerful in dealing with emerging bandwidth hogging applications.

Another application would be in routing situations where traffic on multiple interfaces can be managed in different ways. Connection oriented limits on the private interface can be supplemented by a different type of rate limit on the public interface. Where Border Manager is used to manage multiple DMZ areas, the TSE can be used to control the flow of traffic between zones.

A further technique would be to use a different rate scheme on inbound and out bound frames even when the frame are of the same type. In the case of an IPXIP Gateway, you could manage the inbound and outbound IPX frames differently by using two sets of rules bound to each half-bind. This also provides a generic means of applying data rates that match asymmetric Internet feeds, like cable modems and DSL connections. It would be silly to implement a symmetric rate when the Internet feed provides 1 MB download and 400 K upload. The TSE easily accommodate this for production and prototyping.

## **Case 10 – Applying data rates at certain times**

In an academic environment you might want to enforce different data rates at different times. The TSE does not *directly* support this. But it can easily be done with a little trickery. You could work up alternate versions of your TSE configuration files, for example SHAPER.AM and SHAPER.PM. Using the NetWare CRON and the Novell TOOLBOX you could unload the TSE, copy the desired configuration into place, and load the TSE using .NCF files. The CRON would run the AM or PM version of the NCF file. The unload and reload of the TSE is pretty harmless and is unlikely to cause any noticeable disruptions. The entire switchover can be completed in less than 10 seconds. Having NCF files to swap configurations is also very handy in general.

Suppose you manage a LAN for an academic institution. During the day you do not want your Napster happy students ( or staff / faculty ??? ) competing with important stuff, like registration. The daytime configuration might limit the total throughput available to students or implement a bandwidth banking limit for all workstations. At night, when mission critical systems won't be impacted, open up the valves. The number of alternate configurations is limitless, as you can create as many scripts and CRON entries as needed.

## **Case N – Many Other Possibilities**

You have seen how the TSE can manage your server's bandwidth in a variety of. While the TSE does have its limitations, numerous other applications exist. Here is a short list:

The TSE can easily be used to simulate WAN connections by rate limiting an interface. For example, if you need to know how your applications would function over a certain speed Frame Relay connection, you can configure the TSE to apply a similar data rate to the LAN interface. Similarly, you could simulate the speed difference between various internet connections by punching in the desire speed and applying the rate to all IP traffic. Even SAT-WAN applications can be fairly accurately simulated using fixed delay options in QoS nodes and rules. It is great for WAN prototyping.

Rates can be excluded or applied or to a limited number of workstations using Rules to identify the workstations by MAC address or IP address. This can be inefficient for large numbers of workstations. But a small number of rules can identify "exceptions" such as

your workstation, wink, wink! Or perhaps you know of a “certain special someone” who needs a *really slow* Internet connection today.

The Beta TSE accommodates up to 5,000 simultaneous connections.<sup>26</sup> This is certainly enough for production testing. The 1.0 release will be capable of managing over 30,000 connections. So it can be used in large sites and on heavily used routers or servers.

The TSE can be used to limit the number of broadcasts the server can see, as well as the number received from any given workstations. If you occasionally have a workstation that screams broadcasts, the TSE can prevent the server from seeing them, avoiding the potential performance and stability issues. This is accomplished by a threshold driven connection-oriented rate applied to incoming broadcast frames.

Relatively accurate load balancing can be achieved for SAP located services by imposing minute delays on incoming broadcasts. By imposing small delays on Get Nearest... requests the NOT-LOGGED-IN connections or unauthenticated NDS connections can be sloughed of onto other servers. This is helpful when trying to schedule down time. Using the TSE a couple days ahead of time to divert unnecessary connections to other servers can minimize the visibility of scheduled downtime to users who do not need access to the box.

All of the above cases can be performed at wire speed on decent server hardware. You can rate limit normal NCP based NetWare traffic just as you would the much smaller amount of Internet traffic you need to manage. This is especially helpful when rogue users saturate the network copying large files. It helps keeps everyone on a level playing field.

---

<sup>26</sup> The number of memory nodes allocated limits the number of simultaneous connections. Each connection requires 2 nodes. About 10,000 nodes are allocated when the TSE loads. The TSE also uses the memory nodes for normal operations, so the actual number of connections the TSE can manage may be somewhat less than 5000. The production release of the TSE will support in excess of 30,000 simultaneous connections.

## **Appendect omies**



## **Appendix A - Disaster Avoidance Features**

The TSE monitors its operations and the state of the server about 20 times a second looking for emergent problems. In addition, the TSE checks the consistency of internal data structures as they are accessed and on a periodic basis. Should a problem be found, the TSE can attempt to recover from the problem, disable or shutdown the TSE, or emit a warning message.

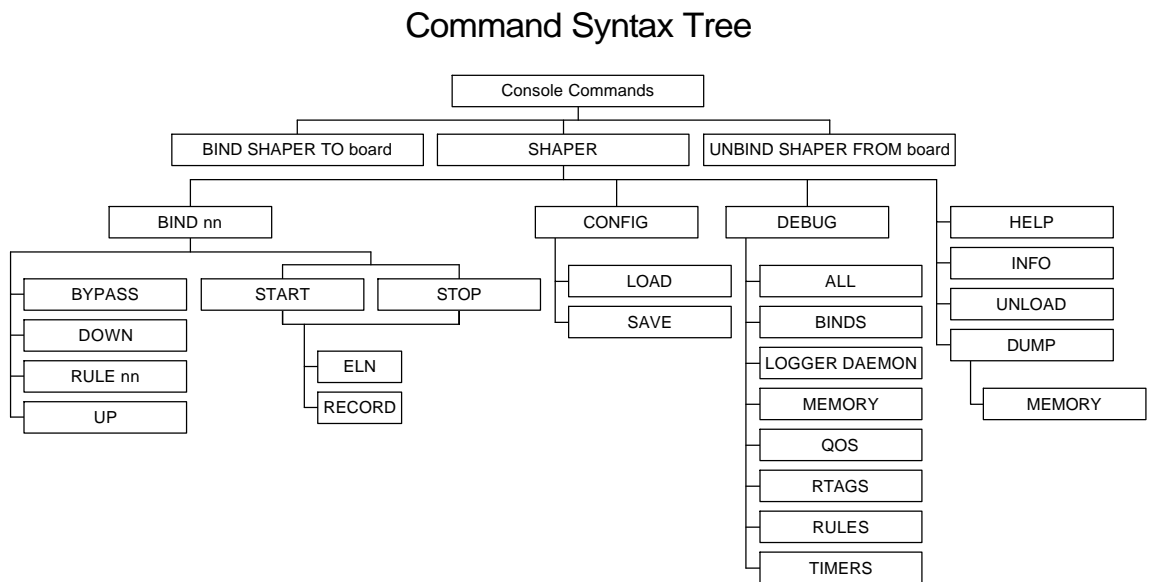
The TSE will spontaneously unload if it discovers the corruption of critical data structures or if abnormalities are detected in certain server data structures.

The TSE will enter bypass mode on all interfaces when it runs low on memory. It will also enter bypass mode when abnormalities in the queuing process are detected. This includes frames being de-queued prior to their ETA or excessive numbers of recoverable errors in queue data structures.

The TSE will automatically recover from corrupted data structures in the queue. The extent of the recovery is to delete and reinitialize lists of en-queued frames. This process is called a "queue zap" and normally never happens unless memory has been corrupted, either by a program bug in the TSE or by another misbehaving NLM. Even a small number of queue zaps is a concern, but are normally recoverable with the loss of only the frames managed by that particular queue structure. Queue zaps are reported on the server console. Unloading and reloading the NLM is advised to free the packet receive buffers that get caught in limbo.



## Appendix B - Command Syntax Tree





## Appendix C - Console Command Reference

### BIND SHAPER TO logical\_board\_name

This is the standard NetWare console command used to bind a protocol to an interface. The logical board name is specified using the NAME= parameter when you load the LAN driver. If you manage your binds manually, you may not be using the NAME= parameter, and so differentiate between binds purely by frame type. This is normally confusing. You should use the INETCFG utility to manage your bindings, INETCFG will give each a unique, standardized name. In any event, you should use the NAME= invocation when loading your LAN drivers. This will allow the TSE to provide a more human readable list of current bindings during operation.

### UNBIND SHAPER FROM logical\_board\_id

This is the standard NetWare console command used to unbind a protocol from an interface. Avoid using bind and unbind the TSE frequently. The TSE will consume a small amount of memory for each bind that cannot be released until the TSE is unloaded. This memory shows up as “toxic” nodes in the TSE’s memory statistics. It is also inadvisable to unbind the TSE during periods of peak activity – this can cause frames to be discarded and may temporarily interfere with network traffic passing through the TSE. The TSE tries to deliver any queued frames for a given logical board, but is unable to do so if the queued frames refer to a board which has been unbound. To avoid this, be sure to put the particular bindings into bypass mode ( see below ) prior to unbinding.

### SHAPER BIND bind\_num RULE rule\_num

This command sets the initial rule to be processed for a given half-bind. This rule must be present in the running configuration. The incoming and outgoing frames can be managed separately by using different rule sets for each half-bind. Setting the rule number to an invalid or undefined rule can cause failure of the TSE.

### SHAPER BIND bind\_num BYPASS

This command tells the TSE to **forward** all traffic for this particular half-bind. When in bypass, the TSE forwards all traffic without examining or queuing it, as though the TSE is not present.

### SHAPER BIND bind\_num DOWN

This command tells the TSE to **drop** all traffic for this particular half-bind. When the half-bind is down, the TSE discards all traffic without examining or queuing it, the TSE acts as a filter. ***This command must be used with extreme caution***, if ever, as it will cause a loss of connectivity for all hosts whose traffic is being dropped.

### SHAPER BIND bind\_num UP

This command tells the TSE to **manage** all traffic for this particular half-bind. When the half-bind is up, the frame is examined by the set of rules starting with the one specified by the SHAPER BIND ... RULE ... command. The default state of a new bind is bypass, you must explicitly use this command to tell the TSE to start managing the traffic. This is best done through a startup script that loads, binds, and configures the TSE.

### SHAPER CONFIG LOAD

This command forces the TSE to reload and activate the contents of the `SYS:SYSTEM\SHAPER.CFG` file. This command is normally not necessary as the TSE will usually detect changes to the configuration file and automatically reload it.

### SHAPER CONFIG SAVE

This command forces the TSE to write the running configuration to the `SYS:SYSTEM\SHAPER.CFG` file. This command is normally not necessary as the TSE will usually detect the deletion of the configuration file and automatically rewrite it. This command may be used to capture runtime statistics and diagnostic information.

### SHAPER DEBUG ALL

This command displays all debug information to the server console. The information will scroll off the screen, so the use of CONLOG to log the console output is often necessary.

### SHAPER DEBUG LOGGER DAEMON

This command displays low level debugging information about the logging facility.

### SHAPER DEBUG MEMORY

This command displays low level debugging information about the TSE's memory usage. The TSE allocates a single large block of memory and manages the block itself. The memory is divided into "nodes" consisting of 64 bytes each. The debug output typically looks like:

```
Memory: AllocAdress=4657010 AllocSize=720384
        FTQMem=465B410 RuleMem=4663810 QoSMem=4667C10 MaskMem=4669010
        HTMem=4657010 NodeMem=466A410 FLCheck=50085036
Nodes:  Orig=10008 Free=9464 Toxic=0 InUse=544
```

The "Nodes:" line shows the number of 64-byte nodes originally available, the number free, and the number in use. The "toxic" nodes are nodes which are no longer needed by the TSE, but for which the NetWare OS *may* still have callback references to. Toxic nodes cannot be freed until the TSE is unloaded. A small number of toxic nodes will accumulate each time the TSE is unbound from an interface. This is normally not a cause for concern.

### SHAPER DEBUG QOS

This command displays low level debugging information about QoS nodes which are part of the static configuration. This command does not display information about QoS nodes that are created dynamically through address masks and connection oriented data rates. This list can be very long so as to scroll off the screen, in such cases use the console logging facility to capture this information.

The output will typically look like this:

```
QoS: 0 Node=4667C10, Flags=0015, Bytes=0Gb + 0, Frames=0
      Period=10000000ns, Rate=0 per 1000000µs, ETA=188136405µs
      Flags: Immortal Fixed ByFrames
QoS: 1 Node=4667C50, Flags=0015, Bytes=28Gb + 12313316, Frames=79148722
      Period=1000ns, Rate=445355 per 1000000µs, ETA=-3772µs
      Flags: Immortal Fixed ByBytes Touched Limiting
QoS: 2 Node=4667C90, Flags=0014, Bytes=0Gb + 0, Frames=0
      Period=10000000ns, Rate=0 per 1000000µs, ETA=188136173µs
      Flags: Fixed ByFrames
QoS: 3 Node=4667CD0, Flags=A009, Bytes=0Gb + 3940, Frames=106
      Period=2000ns, Rate=74 per 1000000µs, ETA=-1071715734µs
      Flags: Immortal ByBytes Touched Limiting
QoS: 10 Node=4667E90, Flags=0008, Bytes=0Gb + 0, Frames=0
      Period=10000ns, Rate=0 per 1000000µs, ETA=188135888µs
      Flags: ByBytes
```

Each QoS node configured in the configuration file will be listed. Consider QoS 1 shows above. The Bytes counter indicates the number of Gigabytes + a byte count. The frame counter indicates the total number of frames. Together this indicates the total volume of traffic managed by the QoS node. In this case a bit more than 28Gb. The Period value is used to calculate the desired data rate. The Period value is roughly the reciprocal of the desire rate, expressed in nanoseconds. The above rate is 1000000 bytes per second, i.e. 1000 ns per byte. The Rate indicates the actual data throughput for the sample period, which is shown to the right of the Rate. The sample period is expressed in microseconds.

For this QoS node the data rate is 443 K bytes per seconds, as  $1000000\mu s = 1$  second. The ETA value is the estimated time of arrival of the last frame processed by the QoS node. A negative ETA means the frame should already have exited the queue. A positive ETA shows the amount of time remaining until the most recent frame processed will exit the queue. If the ETA is more than 8000000, the  $ETA < -35$  minutes, no traffic is being processed. The Touched flag indicated that at least 1 frame was processed by the QoS node, and the Limiting Flag indicates the rate is in effect. When thresholds are in use, the Limiting flag is present only when the rate is “on” and absent when “off.”

### SHAPER DEBUG RTAGS

This command displays low level debug information about the resource tags allocated by the TSE. This information is normally of limited use except when requested for troubleshooting. The output typically looks like:

```
RTags: ECB=7D2D770, Def=7D2D7C0, RT=7D2D630, TX=7D2D810
       Prot=7D2D680, MEM=7D2D6D0, Timer=7D2D720, Poll=5CB3870
```

If any of these are set to 00000000 or contain the value DEADBEEF<sup>27</sup>, there was a problem allocating resource tags. This situation will be caught by the TSE at load time, so that invalid values should never be observed.

### SHAPER DEBUG RULES

This command displays low level debugging information about rule execution. The number of hits and misses for each rule is shown. When a large number of rules are configured, some of the output may scroll off the screen, use the console logging facility to capture this information. The output will typically look like:

```
Rule: 0 Node=4663810, Flags=03120011, Hits=1697022, Misses=22029
      Flags: Process
Rule: 1 Node=4663850, Flags=03110021, Hits=0, Misses=22029
      Flags: Process
Rule: 2 Node=4663890, Flags=11021051, Hits=22029, Misses=0
      Flags: Process
Rule: 3 Node=46638D0, Flags=11120011, Hits=21923, Misses=106
      Flags: Process
```

The number of Hits and Misses are shown for each rule. This information can be used to optimize the execution order of the rules or to simplify rules so there are less to execute. The most limiting rules should be executed first to reduce the number of times addition rules are needed to further discriminate between different types of traffic. In the above listing, it is very suspicious that the number

---

<sup>27</sup> DEADBEEF is used to populate certain TSE pointers and counters prior to initialization of the TSE. The presence of this value means that some initialization routine failed to complete. AbEnd conditions where this value shows up in registers or on the stack indicate a or bug failure during startup.

of Misses from Rule 0, Misses from Rule 1, and Hits for rule 2 are the same. It may be that Rule 1 and Rule 2 can be reduced to a *single* rule or made part of the Else action of rule 0. Performing this type of analysis against your own rules sets after a period of operation is very valuable.

### SHAPER DEBUG TIMERS

This command displays low level debugging information about internal timers maintained by the TSE. This information can be useful when trying to identify performance or loading issues. The output typically looks like:

```
Timers: MFT: CallCount=9795105 State=2
        HRT: Count=582413864, Val=1989793801 LastTick: Count=94338
```

The MFT CallCount indicates the number of times the TSE was called for the servers clock tick interrupt. This value should increase by about 18 a second irrespective of server load. This value should not roll over for 7 years and so can be used as an indicator of TSE up time. The HRT values are not normally significant, as they roll over very quickly. However the LastTick Count indicates the number of times the TSE was called to service the queue during the last clock tick. For the queuing to have decent resolution and accuracy, this value needs to consistently be above 1000. The value of indicates the TSE queuing should be very accurate. When the server is very busy, this value may drop very low. As long as this value is above 1000 on average, the TSE should function normally. The higher the number the better. The value above is from a Pentium II / 300Mhz.

### SHAPER DUMP MEMORY

This command forces the TSE to write a copy of its memory block to a pair of files. The SYS:SHAPER.MEM file contains the raw binary contents of all of the TSE's 64-byte nodes. The SYS:SHAPER.MAP file is a human readable text file listing the offsets into the file and the contents to be found at that offset. This command should only be used for diagnostic purposes and is of limited use otherwise.

### SHAPER OPTIONS

This command lists the state of various options set with the OPTION commands listed below.

#### SHAPER OPTION FORWARD SMALL DELAYS

The queuing mechanism has a resolution of 100Üs. Delay values less than this are treated as zero.<sup>28</sup> This option tells the TSE to forward frames with small delay values rather than put them into the queue. This option can improve the efficiency of large data rates in a LAN environment.<sup>29</sup> See the corresponding QUEUE SMALL DELAYS option below.

#### SHAPER OPTION FORWARD ZERO DELAYS

Internally the TSE “forwards” frames by passing a zero delay value to the queuing mechanism. These frames enter the queue and are released the next time the queue is serviced. This option tells the TSE to *really* forward frames with zero delay values rather than put them into the queue. This option can greatly improve the efficiency of forwarding traffic. *Please feel free to use this option to improve throughput.* The Beta TSE has this option disabled by default to exercise the queuing mechanism. The production release will have this option as a default. See the corresponding QUEUE ZERO DELAYS option below.

---

<sup>28</sup> The queue cannot distinguish between delay values that differ by less than 100Üs, so 230Üs and 290Üs are essentially the same. Normal delay values range from 1000Üs to 500000Üs, so this is not really an issue.

<sup>29</sup> The data rates tend to be larger when managing LAN bandwidth, so the delay values tend to be on the lower end of the range indicated above. Enabling this option lets the TSE forward frames with trivial delays rather than queuing them, this improves throughput under these conditions.

#### SHAPER OPTION QUEUE SMALL DELAYS

This option tells the TSE to queue frames with small delay values rather forward them. This option is the default, but can be modified with the FORWARD SMALL DELAYS option above.

#### SHAPER OPTION QUEUE ZERO DELAYS

This option tells the TSE to queue frames with zero delay values rather forward them. This option is the default, but can be modified with the FORWARD ZERO DELAYS option above.



## **Appendix D – Resolving Performance Issues**

The TSE will generally operate properly on servers with heavy load. This section provides information on how to identify TSE related performance issues and some tricks that can be used to improve the performance of the TSE.

First of all, you need to determine if there is, in fact, a problem. Server utilization, by itself, is not an indicator of poor performance. There are several factors than can impact the overall throughput of a network. It is also the TSE's job to limit bandwidth – which can look to the user to be a performance issue.

For the TSE to function properly, the OS must allow the TSE to service its queues. Under normal conditions, this happens when the CPU is idle. The TSE displays “Queue Stall” or “Granularity Threshold” messages when the TSE is not called frequently. The Queue Granularity Error message means that the TSE was called less than 200 times per second – which severely limits the accuracy of the queue. Under such conditions the observed throughput of the TSE will drop for desired data rates in excess of 100 K per second. The reduce observed throughput is not the result of dropping network traffic, so workstations will function normally.

The Queue Stall Error indicates that the TSE was not provided with any opportunities to manage its queues during the last clock tick. This can be a serious problem since the TSE will dequeue large numbers of pending frames at the next opportunity. While still not a catastrophic problem, the TSE will not be able to accurately impose your desired data rates.

These two errors should not occur very frequently, you may never see them at all on a fast server. However, if these errors occur several times per hour or at times when the source of the high utilization is not obvious, you will need to take measures to analyze and improve server performance. If you get an occasional message there is little to worry about. Both error messages display the number of clock ticks the condition existed, allowing you to gauge the severity of the problem. You will often get these messages when the server is performing certain tasks, such as loading and unloading of NLM's.

Even if you do not receive the above errors, the TSE's queuing mechanism may be barely scraping by. To determine if the TSE is being provided with enough opportunities to service its queues, use the `SHAPER DEBUG TIMERS` command and look at the "Last Tick: Count" value. This value indicates the number of times the TSE was allowed to process its queues during the last clock tick, about an 18<sup>th</sup> of a second. This value should, on average, be at least 1000. If this dips below 10, the TSE will report one of the errors above. Use the console command several times when you believe there are performance issues or when the server is under severe load.

If the "Last Tick: Count" value is consistently high, it is *very unlikely* that the server or the TSE is the bottleneck. You should examine other causes such as improperly configured rules, or rules which cause unexpected results – such as imposing a data rate which is too restrictive. There may also be performance issues with your network infrastructure.

If the above steps show a problem, you should examine server performance information and try to resolve any issues you discover. Common problems include:

One or more of the server's volumes is low on space. You absolutely **MUST** have 20% or more free disk space on each volume, as well as having at least 1000 free blocks. This will prevent the server from aggressively block sub-allocating, which can cause the server to stall for up to 2 seconds each time a directory entry is updated. Expect problems if you are seeing console messages about disk space, like:

```
6-29-00   3:34:41 pm:      SERVER-4.10-3227
          Compressed files are not being committed decompressed on volume CDEES due
          to lack of disk space
```

At least 60% of your server's RAM should be Cache Buffers. This provides the server with enough cache memory to meet the needs of NLM's requesting memory, as well as improved file system performance. One of the best overall indicators of cache performance is the LRU Sitting Time. This value indicates the age of the oldest piece of data in the cache. This value is directly related to the amount of time it takes to completely turn over the contents of the cache – an indicator of how much disk I/O throughput is required. If you have 60 MB in cache buffers and the LRU Sitting Time is 30 seconds, your disks have had to perform at least 60 MB of physical I/O in 30 seconds

– 4 MB a second - that is a lot, even for a fast server! By adding more RAM, you can greatly increase the LRU Sitting Time, and hence cache efficiency. *More RAM is always a great investment for a NetWare server.* Even servers with horribly slow '486s can benefit from additional RAM.

A complete treatise on server performance is outside the scope of this little manual. You should examine Novell's NetWare Performance Tuning information, TID 2943356 and TID 2943472, for specific details of evaluating and correcting performance issues.



## Appendix E – Status Screen

The TSE's status screen is still geared towards providing debugging information. Much of the information provided is not of much importance in the daily care and feeding of the TSE. The following provides information on some of the more useful items.

### LSL Global Statistics

~~~~~

```
- Pkts: TXd = 22576574  RXd = 22862882  Qd TX = 0  Unclaimed = 570
- ECBs: Req = 52180940  Fails = 290  Reused = 0
```

The LSL Global Statistics show the total number of frames processed by LSL. The “Pkts:” line shows the number of frames received and transmitted as well as the number of frames which were unclaimed by protocol stacks. The “ECBs:” line shows the number of times an ECB was request and the number of times the request failed. Increasing values for “Fails” indicates insufficient Packet Receive Buffers, causing incoming frames to be dropped by LSL. A small number of failures are expected as the NetWare OS creates new Packet Receive Buffers. But chronically increasing values should be a concern and will result when the OS is unavailable to create more buffers because of insufficient memory or the Maximum Packet Receive Buffers value has been reached.

### SHAPER Debug Statistics

~~~~~

```
- RX: Count=22828734 NoCTCount=0
- Last: ECB=D18D03A4 Board=1 Chain=D05113C0
- TX: Count=22564550 NoCTCount=0
- Last: ECB=D240F3E4 Board=1 Chain=D0511400
```

The RX: and TX: information display counters and other information for frames passing through the TSE via bindings to the SHAPER protocol. Unlike the LS: statistics which will show all frames seen by LSL, these statistics show only frames passed to the TSE by LSL.. The Count indicates the number of frames processed. The NoCTCount indicates the number of frames passed to the TSE for which the TSE is unable to figure out which binding the frame refers to. This should never happen except when binding or unbinding the TSE, and even then only rarely. Large counts in this field indicate the TSE's internal table of binding information is out of sync with the NetWare OS. This can occur because of a program bug with the TSE, memory corruption, or corruption with NetWare data structures. Unloading and reloading the TSE should correct the problem.

- Conn: Created=23375, Deleted=23273, Active=102

The Conn: line indicates the number of connections created, deleted, and currently managed by the TSE. If you are using Address Masks to manage connection oriented traffic, the Active field shows the number of QoS Nodes / connections spawned by address masks. The other fields can be used to gauge the turn over of connections.

- Hash: Searches=44781163, Compares=79710470, Ratio=1.78

The Hash: line shows the number of searches performed against the connection tables and the number of individual comparison operations needed to complete the searches. The ratio of the two are an indicator of search efficiency. The connection table is implemented as a hash. Ideally, the Ratio should be  $1 + (\# \text{ active connections} \div 1000)$ , but this is rarely achieved. The actual value is typically within  $\pm 1$  of the ideal. If it is larger, the address masks may have been poorly chosen, or the address data causes excessive hash collisions. The above example shows an idea ratio of about 1.1 and the actual of 1.78, this is perfectly acceptable. If the ideal were 2.3 and the observed were 8.9, you might want to take a look at your address masks. If a large number of incoming frames include address keys for which there are a particularly high number of hash collisions, the hash routine works with less efficiency.

Compared to other strategies, hashing has the least level of effort compared to other search techniques such as binary trees. In our case, an ideal hash, which can be almost be achieved by the TSE on MAC and IP addresses, of 1000 connections might take only 2 to 3 comparisons to locate any address key. A binary tree search of the same connection table would take at least 10 comparisons under ideal conditions.

- Queue: In=45393284, Out=36459017 / 45393284 ( 80.32% ), Depth=0

The first Queue: line indicates the number of frames processed by the queueing mechanism. The In field indicates the total number of frames enqueued. The Out field shows the ratio of frames released though the high resolution dequeuing process compared to the total number dequeued. The percentage shows the percentage of dequeued frames released by the highly accurate high resolution dequeuing process. In this case over 80% of the frames were released by this means. There may not always be enough CPU time available to perform high resolution dequeuing. This percentage gauges the accuracy of the queue. A value of 80% is actually rather crummy, this

particular server is a '486, so this is to be expected. The theoretical upper limit is about 97.55%, so any value in the mid 90's is very good. The value is greatly dependent on server load. Inaccuracies between observed and desired data rates should make you examine this value. The lower the value, the less accurate high data rates will be. When in the 90's, data rates up to 1 MB per second will be enforced to with less than 10% deviation.

- Queue: Fails=0, Zaps=0, Tail=328, Head=328

The second Queue: line indicates the number of times the TSE was unable to queue or dequeue frames because of errors in queue data structures or concurrency issues with the queue. The Fails and Zaps fields indicate these number of these errors. A small number is acceptable, large numbers or increasing counts may mean that the TSE is having problems. The Tail and Head field indicate the current tail and head of the queue. These are only important when  $\text{Head} \neq \text{Tail}$ , which should never happen.



## Appendix F – Protocol Info

The following information is provided to help you create Rules and Address Masks. A complete reference guide to packet layouts is outside the scope of this manual. Please consult a decent protocol reference such as BANalyzer by Synapse at <http://www.synapse.de>. They offer one of the most complete references, available on CD-ROM, and a free version via the WWW.

### Protocols In General

There are several popular network protocols, the most significant being TCP/IP and IPX/SPX – at least to NetWare sites. Each protocol has very different characteristics, but share several underlying, fundamental features. Most protocols provide for diagnostics, unreliable, and reliable data transmission. A protocol suite may also include ancillary protocols to handle things like routing and address resolution.

Unreliable methods of delivery are “unreliable” because they offer no built in acknowledgement mechanism. They are connectionless because there is no handshaking between hosts. The hosts send data to a particular socket or port on the target host. Recovery from lost frames is either not needed, the transmission is inherently very reliable as in the case of a LAN, or the application is responsible for implementing such a mechanism. IPX and UDP represent these connectionless modes of communication.

Reliable methods of delivery are “reliable” because they offer built in handshaking, connection setup, acknowledgement, and recovery. An application using such a transmission mode is unaware of any difficulties with the connection unless it is terminated or fails because of unrecoverable errors. SPX and TCP are connection oriented and perform handshaking to establish and break down connections as well as implement error correction and recovery.

## Frame Types & Encapsulation

The offset values provided for frame types below correspond to the RAW offset type used by the TSE. The S-MAC and D-MAC and Frame offset types are shorthand for the corresponding offset values in the frame header. The variety of frame types especially

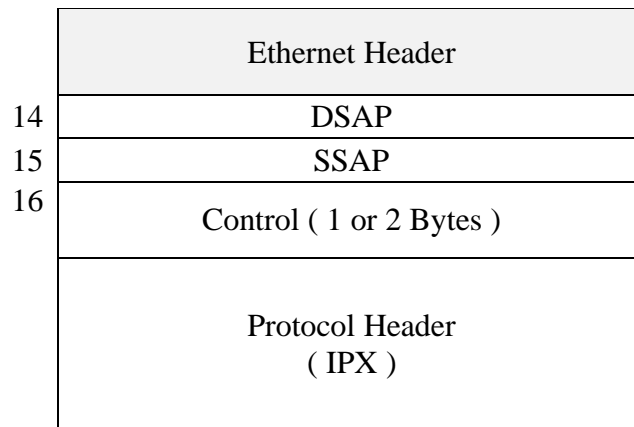
### Ethernet Encapsulation ( Ethernet\_II for IP / 802.3 for IPX )

This is the standard encapsulation for IP and 802.3 IPX frames and is the prefix for LLC and SNAP encapsulation.

	Ethernet Preamble
0	Destination MAC Address
6	Source MAC Address
12	Length / Type
14	Protocol Header ( IP, IPX, ARP, ... )

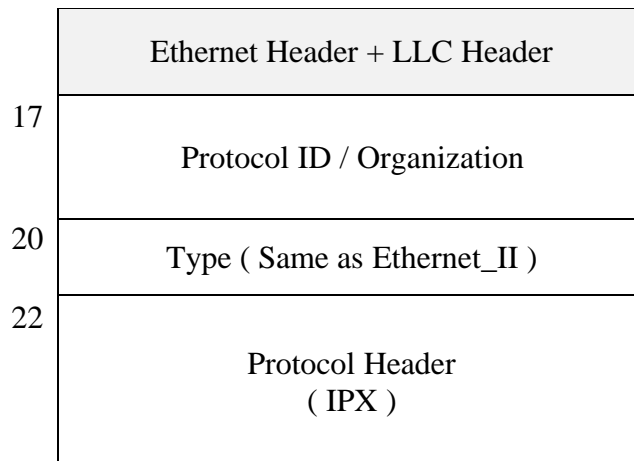
## Ethernet 802.2 Encapsulation ( 802.3 + LLC )

This encapsulation adds a LLC header after the Ethernet header. The offsets shows are from the beginning of the Ethernet frame.



## Ethernet SNAP ( 802.3 + LLC + SNAP )

The LLC header can be 3 or 4 bytes long. However when SNAP is used 3 byte LLC header is present with the LLC header being AA . AA . 03. Cripes!



## IPX

IPX/SPX ( **I**nternet **P**acket **E**xchange / **S**equenced **P**acket **E**xchange ) has long been Novell's primary protocol for connecting NetWare servers and clients. The offset values shown below are Fragment offset values that can be used with the TSE. To examine the source socket number you would use the indicated value of 28 as a Fragment offset.

	Frame Header	
0	Checksum	
2	Length	
4	Transport Control	Packet Type
6	Destination Network	
10	Destination Node	
16	Destination Socket	
18	Source Network	
22	Source Node	
28	Source Socket	
30	DATA ( 1 to 576 Bytes ) e.g. a SPX, PEP, RIP, NCP segment	

## SPX

Add the offset values below to the length of the IPX header ( normally 30 ) to get the fragment offset for the desired SPX field.

IPX Header		
0	Connection Control	Data Stream Type
2	Source Connection ID	
4	Destination Connection ID	
6	Sequence Number	
8	ACK Number	
10	Allocation Number	
12	SPX Payload Data	

## TCP / IP

The TCP/IP protocol suite is the protocol of the Internet and has widely adopted for internal use as well, replacing IPX/SPX for many sites.

Frame Header	
0	Version   IP Header Length
1	Type Of Service
2	Total Length
4	Identification
6	Flags   Fragment Offset
8	Time To Live
9	Protocol
10	IP Header Checksum
12	Source IP Address
16	Destination IP Address
20	Options ( TCP / UDP / Etc. Segments )

## TCP

Add the offset show below to the length of the IP header ( normally 20 ) to arrive at the Fragment offset value to use. For example, the destination TCP port can be accessed with a Fragment offset of  $20$  ( length of the IP header ) +  $2$  ( offset shown below ) =  $22$ .

	IP Header
0	Source TCP Port
2	Destination TCP Port
4	Sequence Number
8	ACK Number
12	Data Offset & Control Flags
14	Window Number
16	Checksum
18	Urgent Pointer
20	Options ( FTP, SMTP, TELNET segments )

## UDP

Add the offset show below to the length of the IP header ( normally 20 ) to arrive at the Fragment offset value to use. For example, the destination UDP port can be accessed with a Fragment offset of  $20 \text{ ( length of the IP header )} + 2 \text{ ( offset shown below )} = 22$ .

	IP Header
0	Source UDP Port
2	Destination UDP Port
4	Length
6	Checksum
8	UDP Payload Data

## Sample Encapsulations

The following illustrate how frames are assembled. They are samples and do not necessarily reflect the encapsulations used in any particular environment.

### Ethernet + IP + TCP

Ethernet Header												IP Header												TCP Header															
Destination				Source				Len						S	Len	ID	FO	T	P	Cks	Source				Dest				Src	Dst	Seq				Ack	DC	Win	Cks	Urg

## The ECB

The ECB data structure consists of several fields used internally by the NetWare OS. The Protocol ID field, shaded below, is particularly valuable. Other fields should not be used.

0	Link
4	Back Link
8	Status
10	ESR Address
14	Logical ID
16	Protocol ID
22	Board Number
26	Immediate Address
32	Driver / Protocol Workspace ( 12 Bytes )
44	Packet Length
50	Fragment Count
52	Rest of ECB Data Structure

## ECB Protocol ID Field Contents

The Table is derived from Novell's *ODI Specification Supplement: Frame Types & Protocol IDs*, available from [developer.novell.com](http://developer.novell.com) or from [www.trafficshaper.com](http://www.trafficshaper.com) web sites. A rule will need to test all 6 bytes of the ECB Protocol ID Field. 802.2 frames may require special processing as described below.

ECB Protocol IDs			
Frame ID	Frame Type	Protocol	Protocol ID
2	ETHERNET_II	IPX/SPX	00.00.00.00.81.37
2	ETHERNET_II	XNS	00.00.00.00.06.00
2	ETHERNET_II	AARP	00.00.00.00.80.F3
2	ETHERNET_II	AppleTalk	00.00.00.00.80.9B
2	ETHERNET_II	ARP	00.00.00.00.08.06
2	ETHERNET_II	RARP	00.00.00.00.80.35
2	ETHERNET_II	IP	00.00.00.00.08.00
3	ETHERNET_802.2	IPX/SPX	00.00.00.00.00.E0
3	ETHERNET_802.2	RPL	00.00.00.00.00.FC
3	ETHERNET_802.2	SNA	00.00.00.00.00.04
3	ETHERNET_802.2	NetBIOS	00.00.00.00.00.F0
5	ETHERNET_802.3	IPX/SPX	00.00.00.00.00.00
10	ETHERNET_SNAP	IPX/SPX	00.00.00.00.81.37
10	ETHERNET_SNAP	XNS	00.00.00.00.06.00
10	ETHERNET_SNAP	AARP	00.00.00.00.80.F3
10	ETHERNET_SNAP	AppleTalk	00.08.00.07.80.9B
10	ETHERNET_SNAP	ARP	00.00.00.00.08.06
10	ETHERNET_SNAP	RARP	00.00.00.00.80.35
10	ETHERNET_SNAP	IP	00.00.00.00.08.00

## 802.2 Frames – LLC Considerations

The 802.2 frame type is rather complicated. There are 3 different header formats, each with different lengths and field layout. On received frames, NetWare sets the Protocol ID field to the DSAP byte padded with zeros: 00.00.00.00.00.DSAP. For transmitted frames the contents of the Protocol ID field tell the MLID which header format to use. The left most byte of the Protocol ID identifies the specific format.

802.2 ECB Protocol ID and Frame Headers										
	ECB Protocol ID Field Contents						802.2 Header Contents			
	0	1	2	3	4	5	1	2	3	4
UI Format	00	00	00	00	00	DSAP	DSAP	SSAP	03	
U Format	02	00	00	DSAP	SSAP	CTL0	DSAP	SSAP	CTL0	
S / I Format	03	00	DSAP	SSAP	CTL0	CTL1	DSAP	SSAP	CTL0	CTL1

On transmitted frames with UI format headers, the Protocol ID field contains a single byte padded on the left with zeros: 00.00.00.00.00.DSAP. This can be seen in the Frame Type & Protocol ID table above. In this case the Protocol ID field would look the same for transmitted and receive frames, greatly simplifying rule construction. This is true for the IPX protocol. The same comparison will work for received or transmitted IPX over 802.2 frames. For IPX over 802.2 use 00.00.00.00.00.E0 for the Protocol ID field.

Other protocols may use multiple 802.2 encapsulation methods. It would be necessary to construct a rule to look for the alternate forms of the Protocol ID value. This is normally not necessary.





## Appendix F - License Agreement

# L I C E N S E   A G R E E M E N T

## Redistribution Or Rental Not Permitted

**These terms apply to  
ShapeShifter Traffic Shaper for Netware  
And / Or  
Beta 1 Traffic Shaping Engine ( TSE )**

BY INSTALLING OR USING THE SHAPESHIFTER OR TSE SOFTWARE (THE "PRODUCT"), THE INDIVIDUAL OR ENTITY LICENSING THE PRODUCT ("LICENSEE") IS CONSENTING TO BE BOUND BY AND IS BECOMING A PARTY TO THIS AGREEMENT. IF LICENSEE DOES NOT AGREE TO ALL OF THE TERMS OF THIS AGREEMENT, LICENSEE MUST NOT INSTALL OR USE THE SOFTWARE.

1. LICENSE AGREEMENT. As used in this Agreement, "The Author" shall mean Robert Charles Mahar, a citizen of the United States Of America. In this Agreement "Licensor" shall mean The Author except under the following circumstances: (i) if Licensee acquired the Product as a bundled component of a third party product or service, then such third party shall be Licensor; and (ii) if any third party software is included as part of the default installation and no license is presented for acceptance the first time that third party software is invoked, then the use of that third party software shall be governed by this Agreement, but the term "Licensor," with respect to such third party software, shall mean the manufacturer of that software and not The Author. With the exception of the situation described in (ii) above, the use of any included third party software product shall be governed by the third party's license agreement and not by this Agreement, whether that license agreement is presented for acceptance the first time that the third party software is invoked, is included in a file in electronic form, or is included in the package in printed form. If more than one license agreement was provided for the Product, and the terms vary, the order of precedence of those license agreements is as follows: a signed agreement, a license agreement available for review on the TrafficShaper website, a printed or electronic agreement that states clearly that it supersedes other agreements, a printed agreement provided with the Product, an electronic agreement provided with the Product.

2. LICENSE GRANT. Licensor grants Licensee a non-exclusive and non-transferable license to reproduce and use for personal or internal business purposes the executable code version of the Product, provided any copy must contain all of the original proprietary notices. This license does not entitle Licensee to receive from The Author hard-copy documentation, technical support, telephone assistance, or enhancements or updates to the Product. Licensee may not customize the Product under any circumstances nor by any means. Licensee may not redistribute the Product unless Licensee has separately entered into a distribution agreement with The Author.

3. RESTRICTIONS. Except as otherwise expressly permitted in this Agreement, Licensee may not: (i) modify or create any derivative works of the Product or documentation, including translation or localization; (ii) decompile, disassemble, reverse engineer, or otherwise attempt to derive the source code for the Product (except to the extent applicable laws specifically prohibit such restriction); (iii) redistribute, encumber, sell, rent, lease, sublicense, or otherwise transfer rights to the Product; (iv) remove or alter any trademark, logo, copyright or other proprietary notices, legends, symbols or labels in the Product; or (v) publish any results of benchmark tests run on the Product to a third party without The Author's prior written consent.

4. FEES. There is no license fee for the Product. If Licensee wishes to receive the Product on media, there may be a small charge for the media and for shipping and handling. Licensee is responsible for any and all taxes.

5. TERMINATION. Without prejudice to any other rights, Licensor may terminate this Agreement if Licensee breaches any of its terms and conditions. Upon termination, Licensee shall destroy all copies of the Product.

6. PROPRIETARY RIGHTS. Title, ownership rights, and intellectual property rights in the Product shall remain in The Author and/or his suppliers. Licensee acknowledges such ownership and intellectual property rights and will not take any action to jeopardize, limit or interfere in any manner with The Author's or his suppliers' ownership of or rights with respect to the Product. The Product is protected by copyright and other intellectual property laws and by international treaties. Title and related rights in the content accessed through the Product is the property of the applicable content owner and is protected by applicable law. The license granted under this Agreement gives Licensee no rights to such content.

7. DISCLAIMER OF WARRANTY. THE PRODUCT IS PROVIDED FREE OF CHARGE, AND, THEREFORE, ON AN "AS IS" BASIS, WITHOUT WARRANTY OF ANY KIND, INCLUDING WITHOUT LIMITATION THE WARRANTIES THAT IT IS FREE OF DEFECTS, MERCHANTABLE, FIT FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PRODUCT IS BORNE BY LICENSEE. SHOULD THE PRODUCT PROVE DEFECTIVE IN ANY RESPECT, LICENSEE AND NOT LICENSOR OR ITS SUPPLIERS OR RESELLERS ASSUMES THE ENTIRE COST OF ANY SERVICE AND REPAIR. IN ADDITION, THE SECURITY MECHANISMS IMPLEMENTED BY THE PRODUCT HAVE INHERENT LIMITATIONS, AND LICENSEE MUST DETERMINE THAT THE PRODUCT SUFFICIENTLY MEETS ITS REQUIREMENTS. THIS DISCLAIMER OF WARRANTY CONSTITUTES AN ESSENTIAL PART OF THIS AGREEMENT. NO USE OF THE PRODUCT IS AUTHORIZED HEREUNDER EXCEPT UNDER THIS DISCLAIMER.

8. LIMITATION OF LIABILITY. TO THE MAXIMUM EXTENT PERMITTED BY APPLICABLE LAW, IN NO EVENT WILL LICENSOR OR ITS SUPPLIERS OR RESELLERS BE LIABLE FOR ANY INDIRECT, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF OR INABILITY TO USE THE PRODUCT, INCLUDING, WITHOUT LIMITATION, DAMAGES FOR LOSS OF GOODWILL, WORK STOPPAGE, COMPUTER FAILURE OR MALFUNCTION, OR ANY AND ALL OTHER COMMERCIAL DAMAGES OR LOSSES, EVEN IF ADVISED OF THE POSSIBILITY THEREOF, AND REGARDLESS OF THE LEGAL OR EQUITABLE THEORY (CONTRACT, TORT OR OTHERWISE) UPON WHICH THE CLAIM IS BASED. IN ANY CASE, LICENSOR'S ENTIRE LIABILITY UNDER ANY PROVISION OF THIS AGREEMENT SHALL NOT EXCEED IN THE AGGREGATE THE SUM OF THE FEES LICENSEE PAID FOR THIS LICENSE (IF ANY) AND FEES FOR SUPPORT OF THE PRODUCT RECEIVED BY THE AUTHOR UNDER A SEPARATE SUPPORT AGREEMENT (IF ANY), WITH THE EXCEPTION OF DEATH OR PERSONAL INJURY CAUSED BY THE NEGLIGENCE OF LICENSOR TO THE EXTENT APPLICABLE LAW PROHIBITS THE LIMITATION OF DAMAGES IN SUCH CASES. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OR LIMITATION OF INCIDENTAL OR CONSEQUENTIAL DAMAGES, SO THIS EXCLUSION AND LIMITATION MAY NOT BE APPLICABLE. THE AUTHOR IS NOT RESPONSIBLE FOR ANY LIABILITY ARISING OUT OF CONTENT PROVIDED BY LICENSEE OR A THIRD PARTY THAT IS ACCESSED THROUGH THE PRODUCT AND/OR ANY MATERIAL LINKED THROUGH SUCH CONTENT.

9. EXPORT CONTROL. Licensee agrees to comply with all export laws and restrictions and regulations of the United States or foreign agencies or authorities, and not to export or re-export the Product or any direct product thereof in violation of any such restrictions, laws or regulations, or without all necessary approvals. As applicable, each party shall obtain and bear all expenses relating to any necessary licenses and/or exemptions with respect to its own export of the Product from the U.S. Neither the Product nor the underlying information or technology may be downloaded or otherwise exported or re-exported (i) into Cuba, Iran, Iraq, Libya, North Korea, Sudan, Syria or any other country subject to U.S. trade sanctions covering the Product, to individuals or entities controlled by such countries, or to nationals or residents of such countries other than nationals who are lawfully admitted permanent residents of countries not subject to such sanctions; or (ii) to anyone on the U.S. Treasury Department's list of Specially Designated Nationals and Blocked Persons or the U.S. Commerce Department's Table of Denial Orders. By downloading or using the Product, Licensee agrees to the foregoing and represents and warrants that it complies with these conditions.

If the Product is identified as being not-for-export (for example, on the box, media or in the installation process), then, unless Licensee has an exemption from the United States government, the following applies: EXCEPT FOR EXPORT TO CANADA FOR USE IN CANADA BY CANADIAN CITIZENS, THE PRODUCT AND ANY UNDERLYING ENCRYPTION TECHNOLOGY MAY NOT BE EXPORTED OUTSIDE THE UNITED STATES OR TO ANY FOREIGN ENTITY OR "FOREIGN PERSON" AS DEFINED BY U.S. GOVERNMENT REGULATIONS, INCLUDING WITHOUT LIMITATION, ANYONE WHO IS NOT A CITIZEN, NATIONAL OR LAWFUL PERMANENT RESIDENT OF THE UNITED STATES. BY DOWNLOADING OR USING THE PRODUCT, LICENSEE AGREES TO THE FOREGOING AND WARRANTS THAT IT IS NOT A "FOREIGN PERSON" OR UNDER THE CONTROL OF A "FOREIGN PERSON."

10. HIGH RISK ACTIVITIES. The Product is not fault-tolerant and is not designed, manufactured or intended for use or resale as on-line control equipment in hazardous environments requiring fail-safe performance, such as in the operation of nuclear facilities, aircraft navigation or communication systems, air traffic control, direct life support machines, or weapons systems, in which the failure of the Product could lead directly to death, personal injury, or severe physical or environmental damage ("High Risk Activities"). Accordingly, Licensor and its suppliers specifically disclaim any express or implied warranty of fitness for High Risk Activities. Licensee agrees that Licensor and its suppliers will not be liable for any claims or damages arising from the use of the Product in such applications.

11. U.S. GOVERNMENT END USERS. The Product is a "commercial item," as that term is defined in 48 C.F.R. 2.101 (Oct. 1995), consisting of "commercial computer software" and "commercial computer software documentation," as such terms are used in 48 C.F.R. 12.212 (Sept.1995). Consistent with 48 .F.R. 12.212 and 48 C.F.R. 227.7202-1 through 227.7202-4 (June 1995), all U.S. Government End Users acquire the Product with only those rights set forth herein.

12. MISCELLANEOUS. (a) This Agreement constitutes the entire agreement between the parties concerning the subject matter hereof. (b) This Agreement may be amended only by a writing signed by both parties. (c) Except to the extent applicable law, if any, provides otherwise, this Agreement shall be governed by the laws of the Commonwealth of Pennsylvania, U.S.A., excluding its conflict of law provisions. (d) Unless otherwise agreed in writing, all disputes relating to this Agreement (excepting any dispute relating to intellectual property rights) shall be subject to final and binding arbitration in Lehigh County, Pennsylvania, under the auspices of Lehigh County Court of Common Pleas, with the losing party paying all costs of arbitration. (e) This Agreement shall not be governed by the United Nations Convention on Contracts for the International Sale of Goods. (f) If any provision in this Agreement should be held illegal or unenforceable by a court having jurisdiction, such provision shall be modified to the extent necessary to render it enforceable without losing its intent, or severed from this Agreement if no such modification is possible, and other provisions of this Agreement shall remain in full force and effect. (g) The controlling language of this Agreement is English. If Licensee has received a translation into another language, it has been provided for Licensee's convenience only. (h) A waiver by either party of any term or condition of this Agreement or any breach thereof, in any one instance, shall not waive such term or condition or any subsequent breach thereof. (i) The provisions of this Agreement which require or contemplate performance after the expiration or termination of this Agreement shall be enforceable notwithstanding said expiration or termination. (j) Licensee may not assign or otherwise transfer by operation of law or otherwise this Agreement or any rights or obligations herein except in the case of a merger or the sale of all or substantially all of Licensee's assets to another entity. (k) This Agreement shall be binding upon and shall inure to the benefit of the parties, their successors and permitted assigns. (l) Neither party shall be in default or be liable for any delay, failure in performance (excepting the obligation to pay) or interruption of service resulting directly or indirectly from any cause beyond its reasonable control. (m) The relationship between Licensor and Licensee is that of independent contractors and neither Licensee nor its agents shall have any authority to bind Licensor in any way. (n) If any dispute arises under this Agreement, the prevailing party shall be reimbursed by the other party for any and all legal fees and costs associated therewith. (o) If any professional services are being provided by The Author, then such professional services are provided pursuant to the terms of a separate Professional Services Agreement between The Author and Licensee. The parties acknowledge that such services are acquired independently of the Product licensed hereunder, and that provision of such services is not essential to the functionality of such Product. (p) The headings to the sections of this Agreement are used for convenience only and shall have no substantive meaning. (q) Licensor may use Licensee's name in any customer reference list or in any press release issued by Licensor regarding the licensing of the Product and/or provide Licensee's name and the names of the Product licensed by Licensee to third parties.

13. LICENSEE OUTSIDE THE U.S. If Licensee is located outside the U.S., then the provisions of this Section shall apply. (i) Les parties aux presentes confirment leur volonte que cette convention de meme que tous les documents y compris tout avis qui s'y rattache, soient rediges en langue anglaise. ( translation: "The parties confirm that this Agreement and all related documentation is and will be in the English language.") (ii) Licensee is responsible for complying with any local laws in its jurisdiction which might impact its right to import, export or use the Product, and Licensee represents that it has complied with any regulations or registration procedures required by applicable law to make this license enforceable.

• • This indicate a clarification to common questions.

! This indicates an important warning which will help you avoid problems, issues, or common pitfalls.

0 This indicates a caveat specific to the beta version of the software. This limitation will most likely not apply to the production release of the software.