

# Dynamic Bandwidth Throttling

*A Technical White Paper*

Chris Foley  
Senior Technical Product Manager  
XcelleNet, Inc.

**XCELLENET™**

## Background

The concept of bandwidth throttling is not new. For over a decade, network infrastructure hardware and software vendors have offered a variety of products to allow Local Area Network (LAN) and Wide Area Network (WAN) administrators to control and manage the traffic traversing their networks. Such products have traditionally been implemented as part of the overall network infrastructure. These products usually operate at Layer 3 or Layer 4 of the ISO protocol stack. Either incorporated into the feature set of **routers** and **switches**, or as stand-alone devices or servers, these products have a scope/jurisdiction of the network segments with which they share interfaces.

Being implemented at such a low level, these devices have limited capabilities when it comes to shaping the network traffic passing through them. A router or switch which allows traffic traveling on a specific port to be restricted to a specified percentage of the aggregate throughput of the device is an example of this type of limitation. In situations where only a single traffic class is routed over a specific port, or to a specific subnet, these products provide adequate distribution of available bandwidth to the respective traffic types or hosts.

## Traditional Bandwidth Throttling and Traffic/Packet Shaping

As network traffic patterns evolved, it became clear that a simplistic port-by-port or subnet-by-subnet approach was not sufficient. Many applications began to “share” the same ports and protocols; HTTP and Port 80 are classic examples. Additionally, as users became more mobile and roving laptop computers started to displace anchored desktop machines, the ability to mediate bandwidth demands by subnet became problematic in many environments. The need developed for a more intelligent means of identifying and throttling specific traffic types, classes, and hosts.

These products, known today as Traffic or Packet Shapers, were introduced to meet these challenges. Traffic Shapers have the ability to better identify application-specific traffic by reading data higher up the protocol stack from within packet headers, and looking for tell-tale application signatures. Armed with this additional data, packet shapers provide WAN/LAN administrators with the ability to control the precise bandwidth percentage used by each respective application.

This top-down view method of bandwidth control is often accomplished by brute-force methods such as packet discardment. For example, if application “A” is generating 15% more packet traffic than it has been allocated, an arbitrary number of its packets are “dropped” or discarded by the packet shaper. This produces the desired effect of *throttling* the application back to target levels, but it does not inhibit the source applications from continuing to generate excessive traffic. In fact, discarding packets usually results in elevated levels of application traffic generation, as negative acknowledgements (NACKS) from remote hosts that frequently generate re-transmissions.

Another limitation of the top-down packet shaping approach is that even under the best of circumstances, the balancing of network traffic only takes the server-host environment into consideration. The experience and performance of remote hosts (and users) is by definition considered to have a lesser priority. The LAN/WAN administrator achieves his or her desired result, but at what cost to the end user?

## Software-Specific Approaches and ‘Static’ Bandwidth Throttling

In response to such network-level approaches, several software vendors have attempted to allow their applications to either “play nicely” in hardware-throttled environments, or to

independently control their traffic demands by providing *throttling* mechanisms. These mechanisms vary in terms of point/location of application, and the methods used. Typically, client applications, such as Web browsers, are at the whim of server throttling settings and fall in line accordingly. Client-server applications, on the other hand, usually allow throttling settings to be configured on the client, on the server, or on both.

As mentioned above, there are several *methods* of throttling employed by differing types of software applications. For the purpose of this discussion, our focus will be confined exclusively to throttling methods used by client-server applications. Because of their dual operating environments, client-server applications are uniquely qualified to overcome the limitations imposed by top-down, server-side only approaches. Despite these inherent advantages, many client-server applications base their throttling methods on arbitrary rules and data. The vast majority of applications that incorporate such an approach can be generally categorized as using a *Static Bandwidth Throttling* model.

*Static Bandwidth Throttling* (SBWT) is similar conceptually to the hardware-based packet-shaping approach in that it targets a specific level of packet activity as its desired result. A typical SBWT implementation would involve either a client-side or server-side setting that gates the amount of packet traffic generated or accepted at pre-configured (*static*) levels. Often these static levels are defined in terms of a percentage of available bandwidth (size of *the pipe*). In most cases, however, the pipe itself must be statically configured or declared. In other words, there is no real-time discovery of available bandwidth, and consequently, no reaction based on availability.

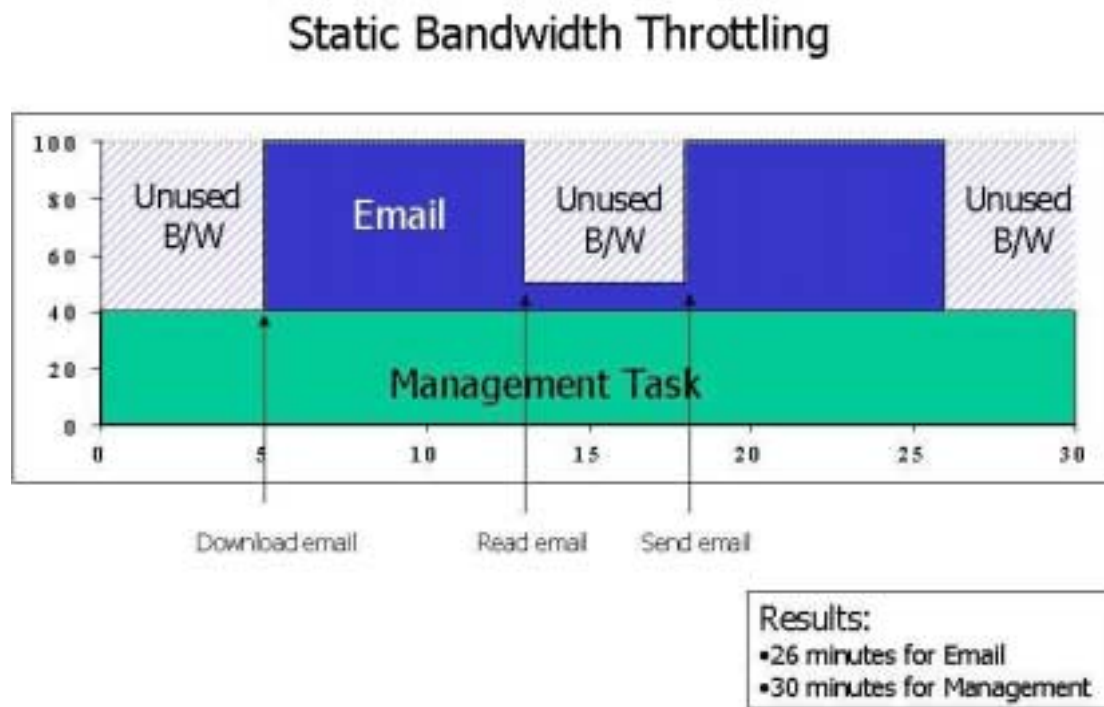
Static Bandwidth Throttling *does* take into account the remote host (end user) experience to the extent that it can be configured to stay within acceptable boundaries relative to other anticipated traffic types on the host. The theoretical best case for this type of bandwidth sharing would be a scenario where each client-side application had a similar facility for gating bandwidth, and the sum total of all applications would be 100% of the pre-determined pipe size (theoretical maximum throughput of the network interface). For example, a remote notebook computer equipped with a 56 kbps modem would be configured to allot 50% (28k) to an e-mail application, 25% (14k) to Sales Force Automation application, and 25% (14k) to a systems management application.

There are many flaws inherent in the **SBWT** model: (a) The theoretical maximum throughput (56k in this case) is rarely, if ever, achieved; (b) Application usage is frequently sporadic, and almost never involves concurrent usage of all production applications; (c) Many remote and mobile computing devices are equipped with multiple network interfaces, such as 56k modems, 10/100 Mb LAN and/or 11Mb WLAN network interfaces (not to mention the growing number of WAN wireless interfaces). Depending on which interface is in use, statically configured throttling parameters may or may not be appropriate; (d) Available bandwidth often fluctuates throughout the course of a communication session, once again rendering static configuration settings inappropriate.

### **Another Static Software Throttling Approach – Choice of Static Parameters**

In an attempt to provide some flexibility in their approach, some software vendors provide a choice of multiple throttling levels. *High*, *medium*, and *low* settings are configurable on the client, the server, or on both. While this scheme represents an improvement over the one-size-fits-all **SBWT** approach, it is still arbitrary in terms of when a given level is applied. A common implementation of this method would be to assign a *high* priority level (no or minimal throttling) to important work, while assigning a *low* priority level (10% or less of the pipe) to non-time-critical tasks. This model still suffers from most of the traditional **SBWT** flaws in that it still gates throughput at static levels, regardless of the connection type/quality/speed, or the amount concurrent application traffic present at communication time.

The net effect of *any* statically configured ceiling is to lengthen the execution time of the throttled application, while also preventing concurrent applications from achieving maximum throughput. As shown in **Figure 1.1** below, **SBWT** models are also incapable of recovering unused bandwidth. This unused bandwidth is frequently the product of the ebbs and flows of most typical production applications, such as e-mail synchronization, database synchronization, Web browsing, and FTP downloads spawned from Web pages.



*Figure 1.1 Static Bandwidth Throttling Behavior*

Clearly, in order to take advantage of these periods of relative inactivity, a bandwidth-throttling scheme must incorporate some form of activity level monitoring. Unlike traffic shaping appliances and servers, which are able to monitor network interfaces directly, most application-level client-side applications do not have access to throughput and capacity levels of these interfaces. In many cases, special monitoring agents must be resident to gain client-side access to such data, while the interfaces themselves must be configured to a *promiscuous* state. The additional CPU and monitoring overhead required to supply connection and interface data to production applications, such as e-mail, management, and data synchronization is considered to be an undesirable by most enterprise administrators. This type of diagnostic activity is usually reserved for isolated troubleshooting scenarios, where *monitoring agents* and *packet sniffers* are typically employed.

Given these constraints, how can a production-class application address the need for intelligent bandwidth control? The answer to this, and many other related questions, is called *Dynamic Bandwidth Throttling™*. XcelleNet, a mobile infrastructure provider, has developed its own patent-pending mechanism that detects and utilizes lulls in bandwidth, while also allowing high-priority applications and tasks to maximize their bandwidth usage as required.

## Dynamic Bandwidth Throttling™ - An intelligent Software Throttling Approach

XcelleNet has developed a unique approach to solving the problems associated with the variable and concurrent bandwidth demands of production-class enterprise applications. *Dynamic Bandwidth Throttling™ (DBWT)*, introduced in its *XcelleNet's Afaria* product line in February 2002, allows administrators to allocate bandwidth on a per-user, per-connection basis. Attributes such as *quality/speed* of connection, *priority* of current task, *time/cost* of connection, and *presence/absence* of other applications are all taken into account in determining when and how much bandwidth is consumed by XcelleNet's Afaria management/data movement activities.

The patent-pending algorithm used in XcelleNet's **DBWT** leverages several platform attributes, which have been key XcelleNet's Afaria differentiators since its initial launch in 1998. Features such as checkpoint restart, byte-level differencing, data compression and file segmentation have enabled XcelleNet's Afaria to move data and perform management tasks efficiently over a myriad of connection types to wide array of remote and mobile devices. These efficiencies and tolerances form the foundation on which **DBWT** is built. With its real-time client and server state discovery capabilities, XcelleNet's Afaria's business rules engine, *Session Manager*, allows administrators to apply the appropriate bandwidth policies *on the fly*.

### How Does XcelleNet's Dynamic Bandwidth Throttling Work?

In order to achieve such flexibility and efficiency without introducing undesirable overhead into its proven communications architecture, XcelleNet uses a pressure-sensing mechanism that relies on many existing components. XcelleNet's Afaria's *Check Speed* Event and *Connection Speed* variable have long been accessible to administrators as a means of varying task execution and priority based on the current speed (or throughput) of a connection. **DBWT** utilizes this measurement as its *Current Throughput* parameter in its monitoring and throttling calculations. Additionally, a running average, *Average Throughput*, is maintained throughout each XcelleNet's Afaria session. Armed with these two pieces of data, XcelleNet's Afaria is able to sense pressure from outside applications by measuring the sustained *Variance* between these two measurements. As shown in **Figure 1.2** below, a sustained drop-off of in the *Current Throughput* relative to the *Average Throughput* is an indication that an outside application or process is exerting "pressure" on XcelleNet's Afaria.



Figure 1.2 – Dynamic Bandwidth Throttling: Sensing and Throttle-down Behavior

Also depicted in **Figure 1.2** is the *Wait Time* value (in seconds), which determines the duration that the variance must be sustained in order trigger a *Throttle-down* reaction. When the target *Variance* is initially achieved, XcelleNet's Aferia sessions are in a *Pending* status until either (a) the *Wait Time* is reached and the *Variance* sustained (*Throttle-down* condition); or (b) the *Variance* is not sustained through the *Wait Time* (resume monitoring condition). Additionally, the *Throttle-down Percentage* is shown. This attribute determines the magnitude of throttle-down behavior. In the example shown, a variance of 20% is sustained for seven (7) seconds. This results in a *Throttle-down* condition, where a 90% reduction in XcelleNet's Aferia throughput results. The *Variance*, *Wait Time*, and the *Throttle-down Percentage* are all administratively configurable, as shown below in **Figure 1.3**. The remaining **DBWT** administrative settings and monitoring tools are discussed in the *Dynamic Bandwidth Throttling Calibration and Monitoring Tools* section of this document.

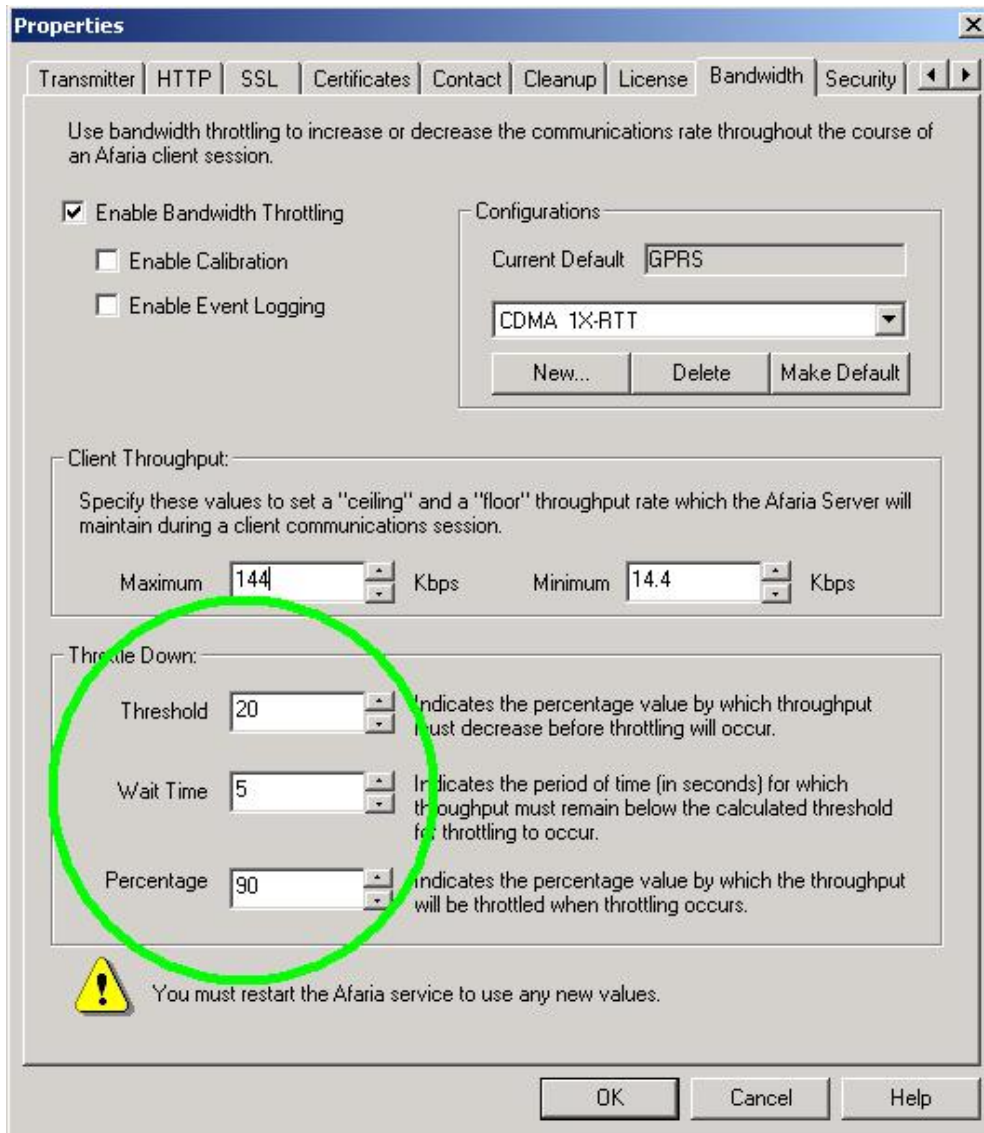


Figure 1.3 – Dynamic Bandwidth Throttling Server Properties Interface

Once a *Throttle-down* action has taken place, XcelleNet's Afaría immediately returns to a monitoring mode. This is the default mode under normal conditions where, over time, XcelleNet's Afaría would eventually drift back to higher throughput levels and eventually fill the pipe (in the absence of outside pressure). However, following a *Throttle-down* condition, XcelleNet's Afaría attempts to accelerate its return to pre-throttle-down throughput levels. It does so by "testing the waters" at three stair-step intervals, which are approximately 10 seconds apart. If throttle-down conditions are once again attained at any of these plateaus, throughput will immediately be throttled-down and the whole cycle will be repeated. This controlled recovery behavior works in a similar fashion to the TCP/IP *slow start* algorithm, and enables XcelleNet's Afaría to take advantage of voids in network traffic. The new state of equilibrium (maximum efficiency) is achieved much faster than if left to natural network drift. This recovery behavior is demonstrated below in **Figure 1.4**.

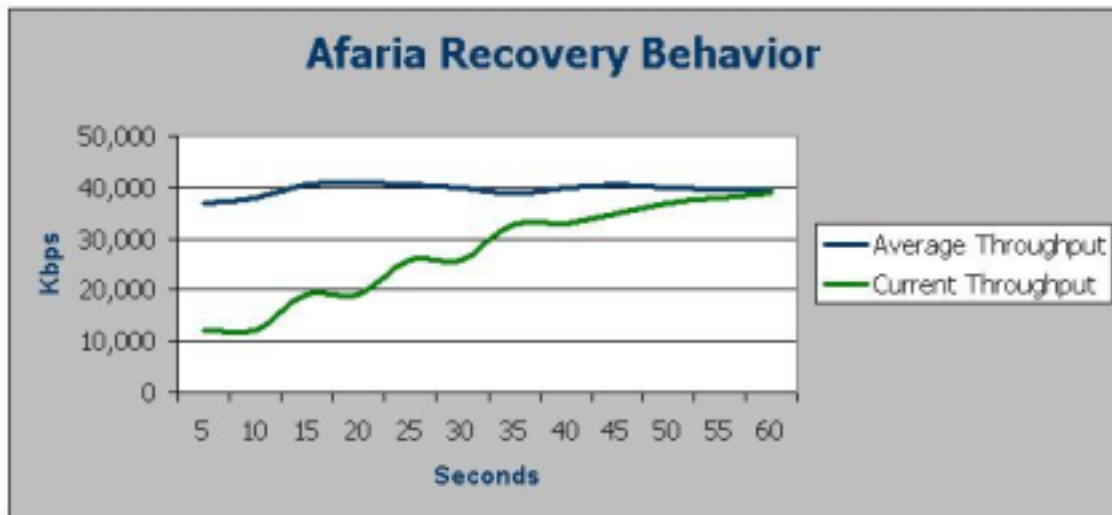


Figure 1.4 – Dynamic Bandwidth Throttling: Recovery Monitoring and Acceleration Behavior

### Tangible Benefits of Dynamic Bandwidth Throttling

By providing a dynamic means of both reducing throughput at times of high contention and aggressively detecting and leveraging lulls in activity, XcelleNet's Afaría is able to *reduce communication costs* while also *reducing end-user pain*. LAN and WAN administrators will immediately recognize the efficiency gain realized by the application of Dynamic Bandwidth Throttling, as average connection times will be reduced. In organizations where the length of connections corresponds to the cost of the connections, the evidence will be show up in their ISP, Telco, and Wireless Carrier invoices. The indirect end-user benefits will show up in a several areas that are not as apparent, yet are just as valuable. By reducing the amount of time that a user spends "on-line" the organization gains a productivity dividend by allowing the user to get back to what he or she is paid to do. Quicker, more-efficient connections also result in greater employee satisfaction levels and fewer Help desk calls. Enterprise employees frequently list "waiting for the organizational stuff to happen" as one of their chief complaints. After all, it's the e-mail, the Web browsing or the sales report that prompted them to initiate the connection, not "that management stuff."

Figure 1.5 below graphically demonstrates the net savings and pain reduction realized with XcelleNet's DBWT. In this depiction, the same amount of work is accomplished in significantly less time due to the inherent efficiencies.

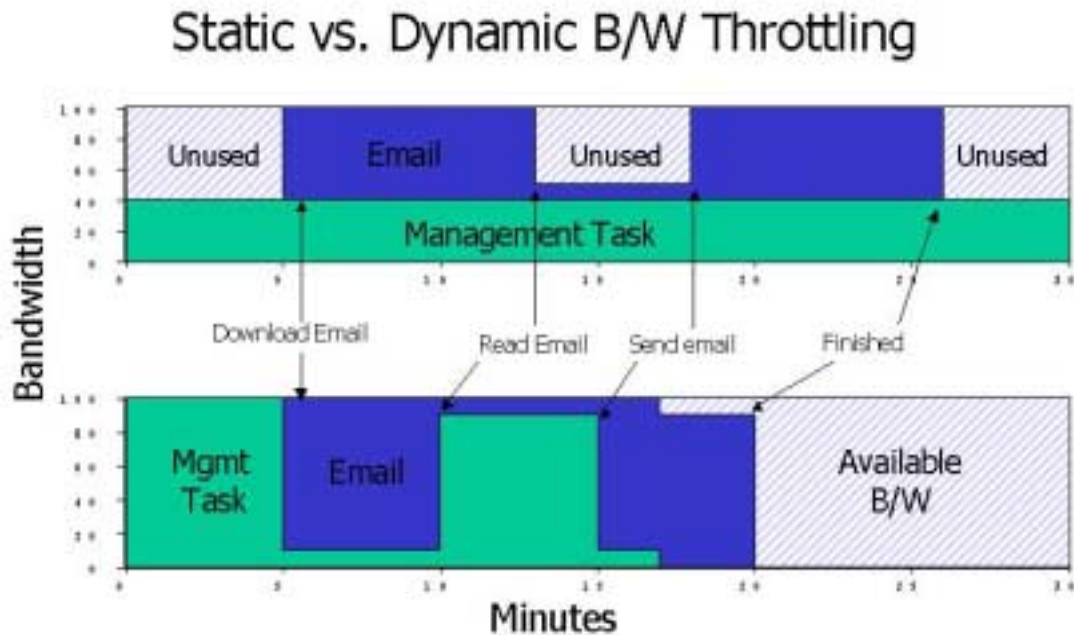


Figure 1.5 – Time, Cost, Productivity, and Support Savings Realized with Dynamic Bandwidth Throttling

### How Does Dynamic Bandwidth Throttling Deal with Differing Connection Types and Speeds and Performance Levels?

While XcelleNet's **DBWT** implementation includes the ability to accelerate and decelerate throughput within a given communication session, the ability to adapt to the type, speed and connection quality of a session is also required. The default throttling settings configured by an administrator may not be appropriate for users connecting over a variety of methods ranging from 9.6 kbps wireless WAN links through 100 Mbps LAN connections. To address these requirements, XcelleNet's Afaia provides several pre-defined sets of throttling parameters appropriate for the most common connection types found in enterprise environments. These "canned" settings can be supplemented with an unlimited number of *custom* settings. Using the real-time-state awareness of XcelleNet's Afaia's Session Manager (is Session Manager trade-marked?), administrators can then create rules to select the appropriate setting on the fly. This capability also supports the ability to change throttling settings multiple times within a single session. By sliding different settings in and out within a session, administrators are able to prioritize critical work, such as delivery of anti-virus updates, while also having the ability to react to fluctuations in connection quality/speed.

The "Configurations" group box shown below in **Figure 1.6** provides a drop-down control listing all pre-defined and custom throttling configuration sets. From this interface, administrators can create new sets, as well as designate the "default" set.



Figure 1.6 – Bandwidth ‘Configurations’ Administrative User Interface

Once Dynamic Bandwidth Throttling parameters are set within the administrative user interface, configuration sets are then applied to XcelleNet’s Afaria Sessions by means of a Session Manager Event named *Set Bandwidth Throttling Config*. This event is usually used in conjunction with procedural logic utilizing system-defined XcelleNet’s Afaria session variables such as *ConnectionSpeed*. **Figure 1.7** (below) depicts a typical scenario where the throughput of the current connection determines which of several possible throttling profiles is to be applied.

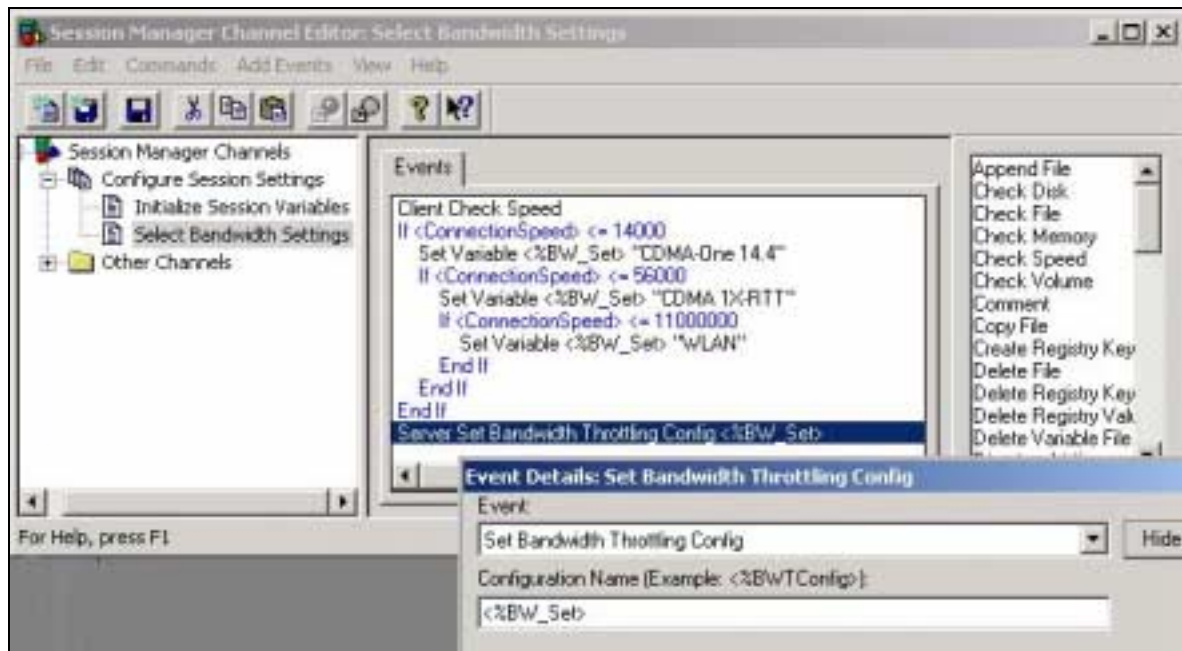


Figure 1.7 – Usage Example of *Set Bandwidth Throttling Config* Session Event

## Dynamic Bandwidth Throttling Calibration and Monitoring Tools

In order to apply throttling configuration sets to actual client environments, the right mix of settings choices must be available. While the pre-built sets provide an excellent starting point, there is no substitute for testing in the production environment. XcelleNet provides several tools to help administrators define, fine tune and monitor their bandwidth throttling configurations. These tools include a set of bandwidth-related *Alerts Console Events*, a *Calibration Mode*, and real-time *Session Monitoring Statistics*.

Two profile-specific settings, *Maximum* and *Minimum Throughput* (see **Diagram 1.8** below), support auditing capabilities that are enabled through the use of the bandwidth throttling *Alerts Console Events*. On the surface, these two profile settings are meant to establish the ceiling and floor between which a given set of throttling parameters operates. Unless the intent is to combine the predictability of **SBWT** with the reactive behavior of **DBWT** within a target range, the value of these settings lies primarily in their ability to test throughput boundaries. Once set at preferred or acceptable recorded levels for each communication type/method, aberrations in performance can then be audited through the XcelleNet's Afaia Alerts Console by defining *Alarms* that are triggered when a designated number of "drops" below the floor or "collisions" with the ceiling occur. Another bandwidth-related Alerts Console Event flags *Throttle-down* instances, which can be aggregated across a specified time period for the purpose of alerting administrators when expected thresholds are exceeded. Checking the "Enable Event Logging" checkbox as shown in **Diagram 1.8** below enables all bandwidth-related Alerts Console Events.

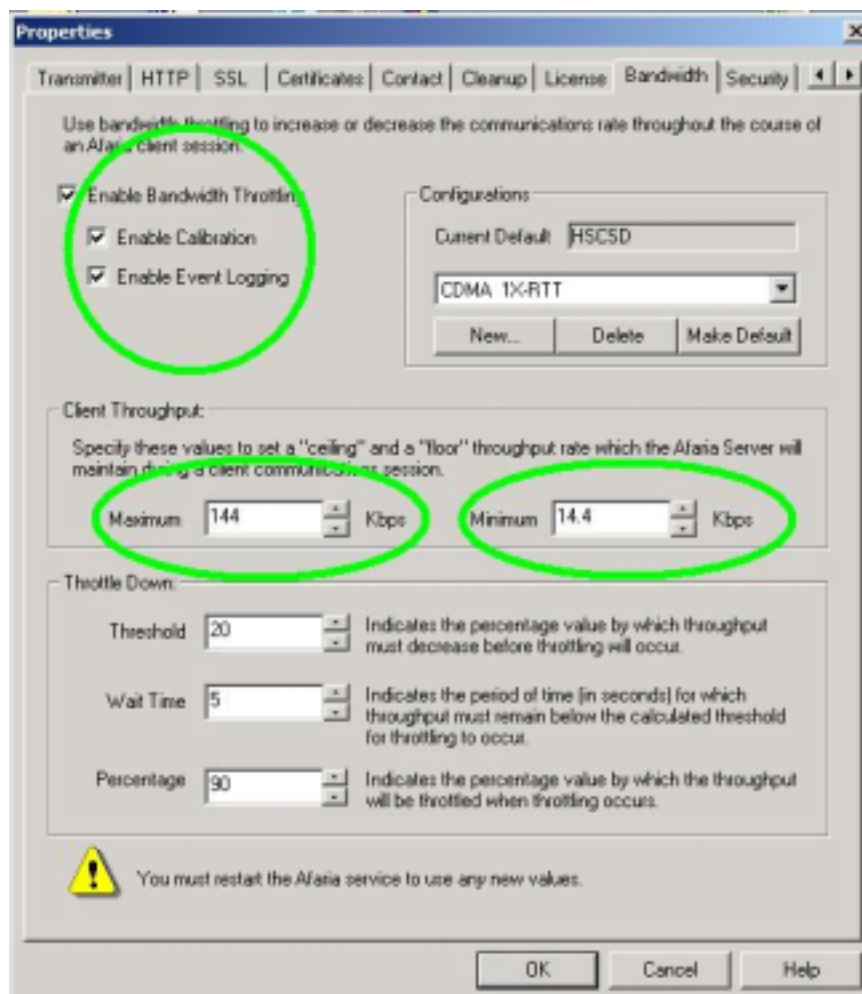


Figure 1.8 – Bandwidth Throttling Calibration, Monitoring, and Range Configuration Settings

Also shown in **Diagram 1.8** (above) is "Enable Calibration" checkbox. When this box is checked/enabled, two separate calibration facilities are activated. Firstly, entering *Calibration Mode* enables administrators to manually or programmatically alter the values contained in custom bandwidth throttling configuration sets without the need to *stop* and *restart* the XcelleNet's Afaia service. The second behavior that is activated in *Calibration Mode* is the generation of an XcelleNet's Afaia Messages Log entry for each client connection, which contains detailed summary information regarding each session's throttling activity. The summary information includes the number of times a *Throttle-down* occurred, the number of times a *Pending* state was reached, and the *Throughput* of the session.

By combining these two Calibration Mode features, administrators can interactively fine tune custom profile settings while monitoring the results in the Messages Log. This is the functional equivalent to turning the “squelch” knob on a Citizens Band (CB) radio until all static noise is suppressed. In other words, administrators can keep “tuning” their settings until they reach a state where no throttling occurs in the absence of other competing traffic or processes. These settings should provide a valid threshold where any additional contention should force a *Pending* condition.

Another monitoring mechanism that provides administrators with real-time feedback regarding **DBWT** performance is the *Session Status* utility. Session Status provides a live view of every XcelleNet’s Afaia Session occurring on a given server. In addition to the basic connection summary information, such as User, Machine Name, Address, Status, Channel and Percentage Complete, this utility provides bandwidth-specific attributes such as *Throttling State, Current Throughput, Average Throughput, Target Throughput, and Threshold*. As with all Session Status attributes, a detailed, Event-by-Event view of the live client connection can be viewed by double-clicking on any selected summary line.

### **Summarizing the Benefits of XcelleNet’s Dynamic Bandwidth Throttling**

By combining the ability to dynamically react to throughput conditions, dynamically change throttling schemes, and dynamically configure and monitor these schemes, XcelleNet’s patent-pending *Dynamic Bandwidth Throttling* implementation provides organizations with a powerful means to **reduce costs** and **minimize end user pain** normally associated with multi-purpose connections. Coupled with XcelleNet’s industry-leading support for enterprise users communicating over slow, intermittent, and unreliable connections, *Dynamic Bandwidth Throttling* takes the control and management of remote and mobile users and devices to unparalleled levels of efficiency and cost reduction.