Causes of slow internet and solut

Technology, Internet

Slow Internet Connection 1. Faulty electrical connection near Internet modem/router Faulty electrical connections can be caused by bad cabling and loose connections. If it's a cable modem, the modem must be attached to the first cable split and preferably the connection with the greatest signal level. If it's a DSL circuit, no other cables should attach to the circuit between the telephone company box and the DSL splitter. . DSL circuit frequency interference If it's a DSL circuit, interference from other circuits in the telephone company's cable bundle can cause packet loss, latency and outages.

T1 circuits are prone for causing problems. T1 circuits installed in neighbors' offices can interfere with DSL. 3. Interference from Wi-Fi radio in Internet Modem/Router Some Internet companies supply modems and routers with built-in Wi-Fi access point capabilities. The built-in access point can be convenient, but it can play havoc with the modem/router itself. Wi-Fi operates in the 2. 4 Ghz frequency range.

That is the same for many cordless phones and microwaves. Using any 2. 4 Ghz device in close proximity to the Wi-Fi enabled modem/router can overload the receive side of a Wi-Fi radio and could cause many problems including device resets. This is true even if the Wi-Fi is not being used as long as the radio is turned on. Make sure the Wi-Fi radio is turned off in software programming for any Internet modem/router. If Wi-Fi is needed, install a stand alone access point. 4.

Virus, Worm or Trojan on a PCViruses, worms and Trojans can do more than just slow down a PC, they can slow down an entire network. Depending on what they are doing, they can be sending out significant network traffic that can slow down an internal network and give the appearance that the Internet is slow. They can also be sending traffic out to the Internet which would indeed slow down the Internet because they are consuming such a large quantity of bandwidth. It is wise to check PCs for malware anytime a network is slow for no apparent reason. 5. Quality of Service (QoS) Parameters Are Not Set Appropriately for the Available Internet Speed When Voice over Internet Protocol (VoIP) is being used as well as other realtime data applications, many companies install/program QoS parameters into their Internet routers. One field in the configuration is the amount of available bandwidth.

The router must know this in order to know when it is time to employ packet prioritization. It also means that the router will not pass through any more bandwidth then this availability setting even if more is available. If QoS settings are active in router configurations, make sure they are appropriate for the available bandwidth. 6. Congested ISP ISPs frequently become congested during heavy traffic and sometimes during normal traffic use. Congestion can also happen between routers in their network and are especially prevalent in meet points between two ISPs. Detecting ISP congestion requires monitoring the Internet circuit with an ISP Packet Loss and Delay Test Tool.

If the problem is intermittent, then the tool must monitor over an extended period of time such as several days or even a week. 7. ISP Not Delivering Promised SpeedUnless the Internet connection is a fixed circuit such as a T1, the delivered bandwidth from an ISP can vary throughout the day. Typically, the varied rate is due to congestion but it can be other factors as well. Still the rate can frequently be below the expected bandwidth. Speed tests can help determine the delivered bandwidth at a precise moment in time. Be careful, as speed tests results can frequently be misleading and they measure only what is happening at one moment versus an extended period of time.

8. Old Browser Profiles Some Web browsers such as Firefox have profiles that can become bloated and corrupted after a while. Unfortunately, they usually work well enough that you wouldn't know they're corrupted. Create a new profile occasionally---especially when you update to major new versions of the software---and see if that doesn't speed up your browsing. 9. High Bandwidth Usage If you like to download large files such as movies, musicand other multi-gigabyte files---and you like to do all these things simultaneously---you will probably see significant slowdown unless you are running a high bandwidth connection such as business-class Internet such as a T1 connection. 10.

Multiple Users of Bandwidth If you have multiple computers using the same connection, each computer uses some of the connection. The more people using the same connection the slower it will run for each. Worse, with Wi-Fi someone within range of your router may be hijacking your connection and slowing it down as a result 11. Malware Less common although still a danger is malware. If your computer becomes infected with certain malware it can be turned into what is known as a zombie, which means that the computer becomes a slave that is at the whim of whoever controls the malware. That person can use your computer for whatever he wants in the background without you knowing it, such as for sending spam emails. To avoid this, use a dependable antivirus, antimalware and firewall software to keep your computer secure.

How Requests are Processed over the Internet | HTTP request/response path The numeric steps below refer to the steps shown above 1. When the user requests a document, the browser issues the request to the local proxy server. 2. This request first goes through an HTTP request filter. The request may be immediately satisfied (e. g. f the request is for a site that is blocked out, such as advertisements), may be modified (e.

g. header compression), or may be passed through without modification. Determination of filterable requests is based on substring matching of URLs to key strings and running corresponding scripts defined in a configuration file. 3. If a response was not generated immediately, the request is logged by the local profile manager and the user profile is updated. 4. The request is then passed on to the cache manager.

5. The profile manager will create a pre-fetch list based on the usage profile and send it to the pre-fetcher. . Requests which change the profile, specifically URLs which point to HTML pages, are sent to the backbone profile engine to enable backbone aggressive pre-fetches and to update the backbone profiles. Note that the use of an explicit connection to send the profile updates is mainly for ease of implementation. A more efficient

mechanism would be to piggy-back such data on HTTP requests that gets transmitted from the local proxy server to the backbone proxy server. 7.

Periodically, the backbone profile engine returns a list of recommended pages to pre-fetch based on group profiles. This can occur when many users of a particular group visit a particular page. Similar to above, such information can be piggy-backed onto HTTP responses in a more efficient implementation. 8. The recommended URLs are operated on by a function in the HTTP request filter to eliminate URLs that would be filtered (i. e. , we do not want to pre-fetch items that we will filter).

This new list is submitted to the pre-fetcher. 9. The pre-fetcher collates the pre-fetch list and group document pre-fetch recommendations that are found to be not filterable. It then amortizes the pre-fetch requests to the cache manager. 10. If the cache has a fresh copy of the document originally requested, the request is satisfied immediately. 11.

Otherwise, the request is forwarded to the backbone proxy server. 12. The normal HTTP transaction occurs between the backbone cache manager and the WWW server. 13. After retrieval, the document is passed through the backbone HTTP response filter. 14. The response is sent back to the local cache manager, who will cache the document if it is a cacheable item.

It is then sent back to the browser (10). 15. The backbone profile manager maintains individual as well as group profiles. Periodically, it creates a list of recommended group documents and sends it to the local proxy server (7) of each member of the group. As profile updates arrive, it creates a list of documents to pre-fetch based on individual and group usage profiles. The only difference from the local pre-fetch list is that the backbone list is longer (i. e.

, we do more aggressive pre-fetches on the backbone). This list is then submitted to the backbone pre-fetch engine. 16. The backbone pre-fetcher will issue the necessary pre-fetch requests. Causes of Possible Delays The main issues that can adversely affect the performance and scalability of your Web services are: * Incorrectcommunicationmechanism. Currently, there are three main technologies for remoting a method call: Enterprise Services, . NET remoting, and ASP.

NET Web services. The best choice depends upon various factors, including the source and target platforms, whether you need to communicate across an intranet or the Internet, whether you require additional services such as distributed transactions, your security equirements, deployment considerations (such as whether your communication must pass through a firewall), other port limitations, and so on. * Web services. Use Web services to build your services. * Enterprise Services. If you use Web services to build your services, you may still need to use Enterprise Services within your service implementation. For example, you may need it to support distributed transactions or if you want to use object pooling.

* . NET remoting. Use remoting for same-process, cross-application domain communication or for remote communication if you need to integrate with a legacy protocol. If you use remoting, avoid custom proxies, custom sinks, and using contexts. This helps to avoid compatibility issues with future

communication technologies. * Chatty calls. Network round trips to and from a Web service can be expensive.

This issue is magnified if clients need to issue multiple requests to a Web service to complete a single logical operation. * Improper choice of parameters. Your choice of parameters depends upon various factors, such as interoperability, the varying platforms used by the clients, maintainability, the type of encoding format used, and so on. Improper choice of parameters can lead to a number of issues, including increased serialization costs and potential versioning problems for the Web service (for example where a custom type is updated). Where possible, you should use primitive types. If interoperability is an issue, consider using
the XmlElement andXmlDocument types and choose types specific to your application, such as an Employee or Person class. * Serialization.

Serializing large amounts of data and passing it to and from Web services can cause performance-related issues, including network congestion and excessive memory and processor overhead. Improper data transfer strategies for large amounts of data. Selecting an appropriate data transfer strategy — such as using a SOAP extension that performs compression and decompression or offloading data transfer to other services — is critical to the performance of your Web services solution. * Improper choice of encoding format. You can use either literal or SOAP encoding. SOAP encoding involves more SOAP-processing overhead as compared to literal encoding. * Lack of caching or inefficient caching.

In many situations, application or perimeter caching can improve Web services performance. Caching-related issues that can significantly affect Web services performance includefailureto use caching for Web methods, caching too much data, caching inappropriate data, and using inappropriate expiration settings. * Inefficient state management. Inefficient state management design in Web services can lead to scalability bottlenecks because the server becomes overloaded with state information that it must maintain on a per-user basis. Common pitfalls for Web services state management include using stateful Web methods, using cookie container based state management, and choosing an inappropriate state store. The most scalable Web services maintain no state. * Misuse of threads.

It is easy to misuse threads. For example, you might create threads on a per-request basis or you might write code that misuses the thread pool. Also, unnecessarily implementing a Web method asynchronously can cause more worker threads to be used and blocked, which affects the performance of the Web server. On the client side, consumers of Web services have the option of calling Web services asynchronously or synchronously. Your code should call a Web service asynchronously only when you want to avoid blocking the client while a Web service call is in progress. If you are not careful, you can use a greater number of worker and I/O threads, which negatively affects performance. It is also slower to call a service asynchronously; therefore, you should avoid doing so unless your client application needs to do something else while the service is invoked.

* Inefficient Web method processing. A common example of inefficient processing is not using a schema to validate input upfront. This issue can be significant because the Web method may de-serialize the incoming message and then throw exceptions later on while processing the input data. How to Enhance the Service Serialization The amount of serialization that is required for your Web method requests and responses is a significant factor for overall Web services performance. Serialization overhead affects network congestion, memory consumption, and processor utilization. To help keep the serialization overhead to a minimum: XML Compression Compressing the XML payload sent over the wire helps reduce the network traffic significantly. You can implement XML compression by using one of the ollowing techniques: * Use SOAP extensions on the server and client for the compression and decompression of requests and responses.

* Use a custom HTTP module on the server and override the proxy for the Web service on the client. * Use HTTP compression features available in IIS 5. 0 and later versions for compressing the response from the Web services. Note that you need a decompression mechanism on the client. Caching Caching is a great way to improve Web services performance. By reducing the average request time and easing server load, caching also helps scalability. You can cache frequently used data applicable to all users, or you can cache SOAP response output.

You can cache application data by using ASP. NET caching features. You can cache SOAP output by using either the ASP. NET output cache or by employing perimeter caching. When designing a caching strategy for your

Web services, consider the following guidelines: Design Chunky Interfaces to Reduce Round Trips Design chunky interfaces by exposing Web methods that allow your clients to perform single logical operations by calling a single Web method. Avoid exposing properties. Instead, provide methods that accept multiple parameters to reduce roundtrips.

Do not create a Web service for each of your business objects. A Web service should wrap a set of business objects. Use Web services to abstract these objects and increase the chunkiness of your calls. Prefer Message-Based Programming over RPC Style You can design Web services by using either of two programming models: messaging style and RPC style. The RPC style is based on the use of objects and methods. Web methods take object parameters to do the processing, and then return the results. The messaging style does not focus on objects as parameters.

It is based on a data contract (schema) between the Web service and its clients. The Web service expects the message to be XML that conforms to the published data contract. This approach allows you to package and send all parameters in a single message payload and complete the operation with a single call, thus reducing chatty communication. The Web service may or may not return results immediately; therefore, the clients do not need to wait for results. Use Literal Message Encoding for Parameter Formatting The encoded formatting of the parameters in messages creates larger messages than literal message encoding (literal message encoding is the default). In general, you should use literal format unless you are forced to switch to SOAP encoding for interoperability with a Web services platform that does not support the literal format. Prefer Primitive Types for Web Services Parameters There are two broad categories of parameter types that you can pass to Web services: * Strongly typed.

These include . NET types such as double and int, and custom objects such as Employee, Person, and so on. The advantage of using strongly typed parameters is that . NET automatically generates the schema for these types and validates the incoming values for you. Clients use the schema to construct appropriately formatted XML messages before sending them. * Loosely typed. These parameters use the string type.

Note that you should not pass XML documents as string parameters because the entire string then needs to be XML encoded. For example, ; lt; and ; gt; needs to be converted to ; amp; lt and ; amp; gt and so on. Instead, you should use either an XmlElement parameter or implementIXmlSerializable. The latter is the most efficient and works well for large data sizes, regardless of which encoding style you use, you should prefer simple primitive types like int, double, and string for Web services parameters. Use of primitive types leads to reduced serialization and automatic and efficient validation by the . NET Framework. Avoid Maintaining Server State Between Calls Maintaining per-caller state in memory on the server limits scalability because the state consumes server resources.

As an alternative, you can pass state back and forth between the client and Web service. Although this approach enables you to scale your service, it does add performance overhead — including the time taken to serialize, transmit, parse, and de-serialize the state with each call. Input Validation for

Costly Web Methods If you have a Web method that performs costly and time-consuming processing, consider validating the Web method input before processing it. It can be more efficient to accept the validation overhead to eliminate unnecessary downstream processing. However, unless you are likely to receive invalid input frequently, you should probably avoid schema validation due to the significant overhead that it introduces. You need to assess your specific situation to determine whether or not schema validation is appropriate. You can validate input data either by using SOAP extensions or by using separate internal helper methods that your Web methods call.

The advantage of using SOAP extensions is that they permit you to separate your validation code from your business logic. If there is any schema change in the future, the extension can change independently of the Web method. Approach to Caching You can greatly enhance Web services performance by caching data. With ASP. NET Web services, you can use many of the same caching features that are available to ASP. NET applications. These include ASP.

NET output caching, HTTP response caching, and ASP. NET application caching. In common with any caching solution, your caching design for a Web service must consider issues such as how frequently the cached data needs to be updated, whether or not the data is user-specific or application-wide, what mechanism to use to indicate that the cache needs updating, and so on. For more information about caching with Web services, see the " Caching" section later in this chapter. Approaches for Bulk Data Transfer and

Attachments You can use the following approaches to optimize the performance of bulk data transfer: * Chunking. With this approach, you use fixed-size byte arrays to send the data one chunk at a time. * Offloading the transfer.

With this approach, you return a URL from your Web service which points to the file to be downloaded. * Compression. You can use a SOAP extension to compress the SOAP messages before transmitting them. This helps when you are constrained primarily by network bandwidth or latency. To handle attachments, your options include: * WS-Attachments * Base 64 encoding * SOAP Message Transmission Optimization Mechanism (MTOM) Avoid Calling Local Web Services Web services located on the same computer as a client ASP. NET application share the same thread pool with the ASP. NET application.

Therefore, the client application and the Web service share the same threads and other related resources, such as CPU for request processing. Calling a local Web service also means that your request travels through the entire processing pipeline and incurs overhead, including serialization, thread switching, request queuing, and de-serialization. In addition,
the maxconnection attribute of Machine. config has no affect on the connection limit for making calls to local Web services. Therefore, local Web services always tend to give preference to the requests that come from the local computer over requests that come from other machines. This degrades the throughput of the Web service for remote clients