HTTP 1.1 Conformance Issues for IPP

Overview

There are a number of implied conformance requirements for origin servers within the current HTTP 1.1 specification. As anticipated by the IPP working group, a significant portion of the current HTTP 1.1 specification is related to the proper operation of HTTP gateways and/or proxy servers. Very little of the document utilizes the word MUST when applied to origin servers. Throughout this document I will use the term "current specification" to mean RFC 2068.

The current specification lists two types of compliance: conditional and unconditional. To quote from the text of RFC 2068:

> An implementation is not compliant if it fails to satisfy one or more of the MUST requirements for the protocols it implements. An implementation that satisfies all the MUST and all the SHOULD requirements for its protocols is said to be "unconditionally compliant"; one that satisfies all the MUST requirements but not all the SHOULD requirements for its protocols is said to be "conditionally compliant."

The ratio of SHOULDs to MUSTs in the current specification is quite high so I would suggest the following criteria be used for our consideration of HTTP 1.1 compliance. If we envision the environment in which IPP clients and servers interoperate as being "closed", then we can assume that conditional compliance would be enough to satisfy our requirements. If the environment in which IPP servers and clients interoperate is "open", then we should lean towards unconditional compliance.

The terms "closed" and "open" in the above paragraph refer to the operational environment in which IPP servers and clients are interoperating. In a "closed" environment, the HTTP "traffic" that is occuring betweeen IPP clients and servers is dedicated to IPP. In an "open" environment, either the IPP client and/or IPP server is operating in a dual-role as both a generic HTTP client or server, and has knowledge of IPP/HTTP protocol (possibly using application/ipp tags).

The rationale for the above proposal is that in a "closed" environment, we have a priori knowledge of exactly the types of HTTP methods, status codes, and to some extent, MIME tags that we can expect over a particular HTTP connection; and this set of methods, codes, and tags are a subset of HTTP, a subset that would fit quite easily within the realm of "conditional" HTTP compliance. In an "open" HTTP environment, we have less knowledge of the types of traffic (methods, URI types, status codes, etc.) that might have to be handled, and the traditional network protocol guidelines should be met: "Be conservative with what you send, but robust in what you can receive". Therefore, unconditional compliance would give us the best chance at interoperating within "open" environments.

One other question that should be considered if we use HTTP as a mapping for IPP is whether or not IPP is "loosely" coupled to HTTP, or "tightly" coupled. "Loosely" coupled in this context means that we are only using HTTP as a pure transport (which is the method my earlier draft employed), and that the "real" IPP packets are encoded within "application/ipp" entities. In the loosely coupled model, we would define our own separate protocol operations, semantics, and status codes. Using HTTP as a pure transport in the loosely coupled model would allow rapid implementation of IPP over other transports (directly to TCP, SPX, etc.).

In a tightly coupled specification, we would leverage as much of the existing HTTP protocol as possible, extending the semantics of certain HTTP methods to be IPP-specific when applied to IPP URIs, as well as reusing all of the existing status codes and MIME-like packaging characteristics employed by the current HTTP 1.1 specification. In the tightly coupled model, implementations of "dual-role" servers (HTTP/IPP) would be easier to deploy and HTTP/IPP clients might be easier to implement. We are also not "reinventing the wheel" if a particular need of our model is sufficiently addressed by HTTP 1.1. In the tightly coupled model, it might be more difficult to port IPP onto some other application-level transport if the need ever arose.

One idea that might have been overlooked is that we should attempt to create conformance requirements for both IPP servers, as well as clients. The group is currently looking at creating a mapping of the IPP model document directly to HTTP 1.1. It is this goal that this document is trying to address with regards to scope and capability. The 'scope' of the effort being how much resources are needed (manpower, code size, RAM, CPU cycles, etc.) to implement IPP over HTTP. The 'capability' aspect of our requirements asks the question whether or not HTTP meets the transport-requirements or operational requirements implied by the IPP model document.

It is assumed that we currently have three alternatives for the choosing a protocol mapping for IPP 1.0:

1. Use HTTP 1.1 origin server capability for IPP servers; use some subset of HTTP 1.1 methods for IPP clients. In this scenario, we could not use the full capabilities of MIME since HTTP does not support MIME, per the MIME standard.

2. Use an HTTP-like protocol for both IPP servers and clients. We don't say that we're HTTP compliant, but we're so close that if we wanted to construct gateways between IPP and HTTP, the work would be more or less trivial. In this scenario, since we are not HTTP compliant, we could choose to use MIME structures that would not otherwise be supported by a conforming HTTP implementation.

3. We "do our own thing", with no resemblance to anything like HTTP. Our own custom encapsulation, headers, status codes, and protocol operation semantics.

At the IETF Plenary in Memphis, and in subsequent teleconferences in the protocol subgroup, it has been suggested that we avoid specifying multiple protocol mappings for IPP, at least for the first standards effort. Rather, our area directors suggested picking only one mapping and going with that. For this reason, this document assumes that the method for job delivery via RFC 1867 to support existing WEB browsers will not be supported. Instead, existing browsers will have to utilize IPP support built into the underlying operating system environment. New browser technology will either use underlying OS support for IPP, or incorporate IPP client capability directly into the browser.

Rather than echo the current HTTP 1.1 specification with regards to MUST and SHOULD requirements, I would advise the WG to review RFC 2068 for the words MUST and SHOULD, with special attention to where and how I have proposed conditional and unconditional implementations of HTTP for IPP clients and servers.

HTTP Proxies

One aspect of implementing IPP using HTTP that must be decided by the working group would be to take advantage of HTTP proxies. Overall inter- or intranet bandwidth could be reduced by allowing intermediate HTTP proxies to cache responses to IPP attribute requests. It is recognized that some intelligent use of response lifetimes would have to be utilized, but this capability is fully supported by

HTTP 1.1, and can be automatically (dynamically) set on a server-by-server basis. The disadvantage is that IPP servers would have to (possibly) implement more support for different types of headers and/or methods that would be utilized by proxies in between dedicated IPP servers and clients. Servers would also have to manage response lifetimes and possibly include support for content-encodings or other headers that proxies might inject into an HTTP transaction.

The remainder of this document is a briefly paraphrased version of my internet-draft regarding key issues with implementing IPP tightly coupled to HTTP.

HTTP 1.1 Methods

GET

HTTP 1.1 GET methods could be used to obtain attributes for different types of IPP objects. If a GET method is applied to an IPP printer object, then the attributes for the printer object are returned. If the GET method is applied to a job object, then the attributes for the job are returned as the response.

POST

The POST method would be used as a way to create IPP objects (printers, jobs, etc.). Initially, the POST operation would be used to map the “CreateJob” model operation. The POST method could also be used to map the “SendJob” model operation. Using HTTP 1.1 persistent connections, multiple POST operations could be used to efficiently deliver job data.

HEAD

HTTP 1.1 servers are required to implement the HEAD method. This method is often used to inquire as to find out meta-information about resources prior to actually performing a GET operation on the resource. This provides a way for clients to learn as much as possible about an object before actually retrieving the contents of the resource with a subsequent GET operation. The HEAD method, like the GET method, could be used to retrieve meta-information (attributes, etc.) about IPP objects before actually accessing it.

PUT (OPTIONAL)

Using the PUT method with an "Allow:" header can provide IPP clients to create IPP objects and specify what types of methods are allowed on the object.

According to section 5.1.1 of the current specification, only the GET and HEAD methods MUST be supported by servers. For a particular type of URI, only certain methods may be supported. In the event of an unsupported method being received for a particular URI, the server is required to return a status code of 405 (Method not allowed), and include an "Allow:" header field listing exactly which methods are supported for the URI in question.

Lightweight embedded IPP server implementations could be unconditionally compliant by only supporting the GET, HEAD, and POST methods, and returning a status code of 501 when requests are received specifying any other unsupported method. The 501 response should include a "Public:" response header indicating which methods the server does support.

HTTP 1.1 General Headers

Many of the headers specified by RFC 2068 do not have to be supported by general purpose HTTP servers. The following text clarifies what I think IPP should utilize and the HTTP 1.1 conformance issues for each.

Cache-Control:

IPP servers and clients will have to operate within a caching proxy environment. In order to ensure a pure client and server environment between IPP clients and servers, we will have to decide whether or not to take advantage of caching, or prohibit caching of any and all IPP traffic. If we choose to prohibit caching of IPP traffic (to keep things simple), then the "Cache-Control:" general header must be included in all IPP requests and responses. The value for the "Cache-Control:" header directive would be "no-cache". Also, to make sure there are no HTTP 1.0 caching proxies between HTTP 1.1 clients and servers, we must also include the "Pragma:" general header, also specifying "no-cache" as the value.

It is conceivable that there would be some value in caching of attribute requests to IPP printer objects, since in a large environment, these requests might be very frequent. The IPP working group may want to consider the value in caching certain IPP object attribute requests. If caching of certain IPP response data is allowed, then we should also consider the use of the "no-transform" value for the "Cache-Control" directive.

Connection:

The "Connection:" general header could be used by IPP servers or IPP clients to instruct either a remote client or server that the HTTP connection be closed. In HTTP 1.1, persistent connections are the rule, not the exception. If we decide to allow a "SendJob" operation to occur in multiple POST or PUT operations to an IPP server, then persistent connections would be very valuable for enhancing performance of job submission. In this type of multiple POST/PUT operation, the last POST/PUT operation required to deliver the job data could include the "Connection:" header with the value "close" to instruct the server that the connection will be closed after reception of this request.

Content-Coding:

The "Content-Coding:" entity header field specifies how the entity body of a particular message is to be decoded. For HTTP, this is typically a compression encoding so the field would be "gzip" or "compressed". If the IPP working group wants to define a base set of content-codings, then the values for these codings would be specified in IPP messages via the "Content-Coding:" header.

Content-Language:

Like the "Content-Coding:" header, this header also specifies information related to the enclosed entity. The "Content-Coding:" header describes the natural language in which the entity body has been encoded. IPP servers should only return entities in languages that have been "agreed" upon by a particular client in a previously received "Accept-Language:" request header.

Content-Length:

The "Content-Length:" header specifies the size of a message body. IPP clients and servers will use the same algorithm as general-purpose HTTP 1.1 servers for determining the length of IPP messages. From RFC 2068, the description of the Content-Length header:

```
Applications SHOULD use this field to indicate the size of the
   message-body to be transferred, regardless of the media type of the
   entity. It must be possible for the recipient to reliably determine
   the end of HTTP/1.1 requests containing an entity-body, e.g., because
   the request has a valid Content-Length field, uses Transfer-Encoding:
   chunked or a multipart body.
```

Content-MD5

The Content-MD5 header can be used by IPP clients and servers to provide a more robust authentication method than just basic HTTP authentication. The current HTTP 1.1 specification states that use of the MD5 digest authentication is sufficient to protect against accidental modification of the message, but NOT sufficient to protect against malicious attempts to modify the message. The IPP working group should consider as few security mechanisms to provide a higher degree of interoperability between clients and servers. It would seem as if two levels of secure access to IPP objects would suffice: a simple method for moderate to insecure sites where security is not an issue, and a very robust method that is sufficient to meet the needs of sites requiring very high levels of security, including commercial transactions. The simple security could be provided by basic HTTP authentication, and a yet-to-be-decided method (maybe SSL or secure MIME) could be utilized in high security environments. With these two scenarios and methods, IPP clients and servers would not generate messages with the Content-MD5 header.

Content-Type:

The Content-Type header would be used by IPP clients and servers to specify IPP-specific entities. The Content-Type value would be “application/ipp”. IPP clients and servers would also supply a Content-Type modifier “charset”, as part of the applicaion/ipp Content-Type. The “charset” modifier would specify the character set used within the application/ipp entity body.

Date:

The “Date:” header field is currently specified in the HTTP 1.1 document as a MUST header by all compliant implementations. The date format used as the value of this header must be in RFC 1123 format. There is a recent internet draft that has been published that attempts to describe how some embedded, lightweight HTTP server implementations can still be “compliant” even if they don’t contain any realtime clock or time capabilities.

Pragma:

The “Pragma:” directive would only be used by IPP implementations for backwards compatibility with HTTP 1.0 caching proxies. The Pragma header would specify the value “no-cache”, which is understood by HTTP 1.0 proxies to have the same semantics as the HTTP 1.1 “Cache-Control” directive with the value “no-cache”.

Transfer-Encoding:

For HTTP 1.1, the only “Transfer-Encoding” specified is the “chunked” encoding. Since an HTTP connection is “8-bit clean”, the traditional rationale for transfer-encodings (like used in MIME) are unneeded. But when IPP implementations are attempting to send messages for which the total length of the message cannot be determined, then the message should be transferred as either “chunked” or via a multipart message with message boundaries. The current HTTP specification requires that all HTTP 1.1 applications MUST be able to receive and decode the chunked transfer encoding.

HTTP 1.1 Request Headers

Accept:

The Accept: header is used to specify certain media types that a client is willing accept as a result of a request to a server. IPP clients should always specify (at a minimum) application/ipp, text/html, and text/plain.

Accept-Charset:

This header indicates to servers what character sets a client is willing to accept in a response. According to the HTTP 1.1 specification, all clients should be able to support ISO-8859-1.

Accept-Encoding:

Similar to "Accept:", the Accept-Encoding header is sent from client to server to inform the server what types of encoding of responses that the client can handle.

Accept-Language:

IPP clients would send Accept-Language headers in IPP requests to notify IPP servers what type of localization is acceptable to the client.

Authorization:

IPP servers may protect certain types of IPP objects via HTTP basic authentication. If an IPP client has knowledge that a requested resource requires basic authentication, then an appropriate "Authorization:" request header should be included in all IPP requests to the IPP object (URI) in question. The client can also dynamically learn of the authentication requirements for a particular object if the client attempts to access the object without an authentication header. IPP servers that receive un-authenticated requests for IPP objects that require basic authentication would return a status code of 401, which indicates to clients that authentication is required for accessing the requested object.

It is assumed that, for the lifetime of a particular IPP object (URI), that the user's credentials (once successfully validated) will be valid. Therefore, on the first successful authenticated response to a request, IPP clients can cache the user's credentials and reuse these credentials on subsequent requests to the server for this object. Each subsequent request for the IPP object (URI) would include an "Authorization:" header specifying the cached credentials.

From:

The “From:” header contains the internet e-mail address for the human individual that is responsible for the request being generated. The IPP working group has talked about using the “From:” header as a means for some type of authentication or access protection. The current HTTP 1.1 specification states that the “From:” header “SHOULD NOT be used as an insecure method of access protection”. The specification goes on to say that “the interpretation of this field is that the request is being performed on behalf of the user specified by the “From:” header, who accepts responsibility for the operation being performed.”. The following paragraph from RFC 2068 is especially relevant:

```
Note: The client SHOULD not send the From header field without the
```

```
      user's approval, as it may conflict with the user's privacy
      interests or their site's security policy. It is strongly
      recommended that the user be able to disable, enable, and modify
      the value of this field at any time prior to a request.
```

Host:

The Host: field typically comes on a separate line after the HTTP method specification. This field MUST be set by HTTP 1.1 clients with the network location of the specified URI in the method. All internet-based HTTP 1.1 servers MUST respond with a 400 status code to any HTTP 1.1 request message which lacks a “Host:” header. This header is used by newer WEB server sites for so-called "virtual host" access. IPP could utilize this field in some very interesting ways with regards to multiple logical printers serviced by a single IPP/HTTP server.

Proxy-Authorization:

When there is an HTTP 1.1 caching proxy operating in between an IPP client and server, it is possible that certain resources identified by a site administrator might require basic authentication. If an IPP client receives a 407 response to a valid IPP request, the client should format an authorization request back to the requested resource (URI) using the “Proxy-Authorization:” request header. Section 11 of the current HTTP 1.1 specifcation discusses HTTP authorization in detail.

HTTP 1.1 Response Headers

Accept-Ranges:

IPP servers could make use of the "Accept-Ranges:" response header for other purposes than just byte ranges. The "Accept-Ranges:" header includes a parameter that specifies what type of range the server is capable of handling; "bytes" is just one possible value for this field. Other possible values could include "pages", "cost", and other types of range values that would be applicable to printer or print job resources.

Allow:

The "Allow:" entity header field can be returned by IPP servers to notify IPP clients which HTTP methods are allowed to be executed on a particular URI (or IPP object). In the future, we may want to define conformance levels with respect to IPP, wherein some IPP servers implement all possible methods on IPP objects, and other lighter weight IPP servers are restricted in the domain of methods supported on IPP objects. The "Allow:" header permits interoperability between clients and servers of different capabilities. The client can adapt its behavior to the capabilities it learns from a particular server.

Content-Location:

IPP servers can return a Content-Location header that specifies the URI of a job object created with the “CreateJob” operation. IPP clients can also use the Content-Location header to specify the target IPP object (URI) to which a particular IPP operation is to apply.

Expires:

If the working group decides that IPP responses can be cached by intermediate HTTP caching proxies, then appropriate use of the “Expires:” header should specify how long proxies (and possibly) clients can consider the response “valid”. It is understandable that in the case of an embedded IPP/HTTP server that does not have access to a time source, that the “Expires:” header would not be generated. In this case, the embedded server should disable caching of responses using “Cache-Control” headers.

Location:

The “Location:” header would be used by IPP servers to dynamically redirect IPP clients to other URIs that can be contacted for completing the client’s request. The Location header could be used as a replacement for the multiple-URL facility discussed in the early IPP-over-HTTP internet draft. IPP implementations would follow the direction set forth by the current HTTP 1.1 specification:

“for 201 (“Created”) responses, the “Location” is that of the new resource created by the request. For 3xx responses, the location SHOULD indicate the server’s preferred URL for automatic redirection to the resource.”

The term “resource” used in the above paragraph would normally be a URI referencing an IPP job object.

Proxy-Authenticate:

It is possible that, in the presence of caching HTTP 1.1 proxies, that IPP client implementations may have to deal with “Proxy-Authenticate” responses. The “Proxy-Authenticate” response header would be returned as part of a 407 (Proxy Authentication Required) response. (see also Proxy-Authorization request header).

Public:

The “Public:” response header would be used by IPP servers to inform IPP clients what types of HTTP methods are supported by the server. The “Public” response header would typically be used by very lightweight HTTP/IPP server implementations that implement a minimal IPP capability.

Retry-After:

The “Retry-After” response header would be used in tandem with the 503 (Service Unavailable) response code to indicate how long the resource (or service) is to remain unavailable. This could be used by IPP servers to indicate how long a printing service might be unavailable to IPP clients.

WWW-Authenticate:

The WWW-Authenticate response header is used to initiate basic HTTP authentication. If an IPP client receives a 401 (Unauthorized) response to an IPP request, then the response MAY contain a “WWW-Authenticate” header with an appropriate challenge. The next request for this resource formulated by the IPP client should contain an “Authorization” header specifying appropriate credentials.

HTTP 1.1 Status Codes

The status codes recommended by Keith Carter, and in a subsequent document from Bob Herriot seem sufficient for a "closed" implementation of IPP clients and servers.

Other issues

Administrative Framework

The care and feeding of IPP client and server implementations should be taken into account during the design of the protocol. This framework can get very complicated, especially if proxy and security issues are taken into account. Before reaching final consensus on a protocol definition for IPP, the complexity of configuring clients and servers should be weighed appropriately.

Print-By-Reference

There is a recent document, co-authored by Keith Moore, entitled “Definition of the URL MIME External-Body Access-Type” (RFC 2017), that discusses an easy way to support the “Print-by-Reference” capability that has been discussed in the IPP working group. To avoid replicating the text of the RFC, an example of the use of this method would looking something like this:

```
Content-Type: message/external-body; access-type=”URL”;
URL=”http://www.yahoo.com/daily-stock-quotes”
```

This method seems to fit very well with the requirement for “Print-By-Reference”. One aspect of this method is that, if any secure access is to be applied to the retrieval of the external body, that any and all security mechanisms would have to be specified (encoded) somehow within the URL string.