IPP Server Contention Analysis

R. Turner

It seems to me that there is a very finite set of circumstances that can occur during the job submission process from client to server. Note that these circumstances can be looked at from two perspectives; 1) from a purely transport perspective, and 2) from an end-user perspective. In my opinion, we need to define the transport problem first. After which, client software is free to translate certain transport encounters to a suitable user experience as would be expected from a particular platform environment.

Client/Server Transport Issues

It seems to me that there are only four possible scenarios for print job submission transport:

1. The entire job is transmitted by the client and received successfully by the server
2. The print job has begun submission but an indefinite delay has been encountered; note that this could occur after only one byte of the job has been successfully acknowledged, or the entire job except for the last byte has been successfully acknowledged, or anywhere in between.
3. The print job has begun submission but is abnormally terminated before all of the print job is submitted.
4. The server is not accepting connections

I think all of the print job submission conditions that could occur would be mapped into one of the above scenarios. Note that there are numerous reasons that might cause a client-server print job submission to fall into one of the above scenarios, but for the purposes of transport issues, we will postpone discussion of actual status codes.

With regards to actual server contention, scenarios 2,3 and 4 might be labeled as contention-related scenarios.

IPP Transport

With IPP, it is easy to confuse HTTP transport issues with underlying TCP transport issues. Given the wild variations in which IPP servers can be deployed, its easy to come with several combinations of IPP server configurations. This document assumes that IPP client implementations will always be dedicated IPP clients that employ both HTTP and IPP protocols in a monolithic fashion. I have outlined a number of possible configurations that I think will make up the majority of IPP servers that are deployed:

1. Monolithic (dedicated) HTTP/IPP server.
2. Generic HTTP server, with traditional CGI-based IPP server
3. Netscape HTTP server with NSAPI-based IPP server
4. Microsoft IIS HTTP server with ISAPI-based IPP server
5. JVM-capable HTTP server with Java-servlet IPP server

The above configurations include multiple methods for determining the status of a particular IPP client-server connection. Within the context of this document, I am assuming that the IPP working group can only “standardize” end-to-end issues relating to status. This assumption implies ignoring transport-specific status (TCP/IP, HTTP) and concentrating on IPP-specific status codes that can be encapsulated by the IPP protocol. If we can rely on our transport to always deliver IPP responses reliably, then for the purposes of contention handling, I think that transport-specific issues should be left up to individual implementations. Especially in light of the numerous configurations that would have to be dealt with (see above), not to mention the existence of HTTP proxies or other IPP gateways.

Contention Handling for IPP

To allow clients to predictably handle possible IPP server contention, we need to define a set of conditions and/or status codes within the IPP protocol. We also need to define predictable semantics that IPP clients should follow when these status codes are received in response to a job submission request. One or more status code semantics could be applied to other IPP operations as well, in addition to job submission operations.

For each of the operational scenarios that could result from contention (2 through 4) outlined at the beginning of this document, I will attempt to address what type of client-server semantic might be enforced for each one.

Scenario 2: Indefinite Delay during print job submission

In this scenario, a classical "spooling" server could be spooling the job to secondary storage as the job is being received. It is possible that multiple threads of execution within the server are concurrently spooling other client-initiated job submission streams to secondary storage. It is also possible that available secondary storage is temporarily exhausted. It is reasonable to expect that client software (as opposed to the actual client user) would sense this condition if for some reason attempts to send further job data were to "block" for an unusual period of time. With a high-volume IPP server, this type of scenario would not be exceptional, and would therefore be a normal circumstance of job submission to this particular server. The term "block" in this paragraph could be used to connote any of the following cases:

1. An IPP request has been successfully transmitted, but no response has been returned, so the overall "IPP transaction" is not complete.
2. An IPP request has been successfully transmitted, but the server response indicates that some resource has become temporarily unavailable to complete the request.
3. The underlying transport on which we are relying for delivering IPP protocol is temporarily "blocked" due to internal protocol semantics ("flow control", "buffer availability", etc.).

For the purposes of "standardizing" IPP behavior, I think the IPP working group should recognize this scenario and define a particular type of response code that reflects this particular condition. This status code would reflect case #2 above. Case #1 would not require protocol support, but rather client-side configuration of timeout values. Case #3 involves underlying transport issues, which, due to the number of APIs in use, is outside the domain of IPP standardization.

Scenario 3: Abnormal Termination of Job Submission

Usually, this scenario would be a degenerate case of scenario 2, wherein either the client or the server would fail due to some timeout condition or other policy. This "policy" might indicate or enforce how long either the client or server would wait for resources to become available to complete the job submission request. In the client case a timeout would typically be used; if the server decides to abnormally terminate job submission it might be due to the fact that the server has determined that one or more resources required to complete job submission have entered into an "offline" state. At this point, the server decides it cannot maintain client sessions that require one or more of the resources that have entered the "offline" state.

From an IPP clients' perspective, abnormal termination can take on the following forms:

1. Some type of "abort" indication is received from the underlying IPP transport.
2. The client receives a "abort" status code from the IPP server for a particular request

Within the scope of the IPP protocol, the working group can define a particular "abort" status code to reflect abnormal termination of a job submission. Transport-level connection abort sequences are beyond the scope for us to define since these status codes vary by transport API.

Scenario 4: No Job Submission Connections available

I believe this scenario to be the most common case that clients will handle with regards to server contention. In my opinion, recognition of this scenario also implies that contention handling for non-spooling, as well as spooling IPP servers will be the same. I define a non-spooling IPP server as a server that can only accept one job submission request at a time. And further, that no other job submission requests can be accepted until the "current" job completes. The non-spooling IPP server is just the degenerate case of a spooling IPP server, when the spooling IPP server has exhausted available resources to accept any new inbound job submissions.

As suggested by many participants on the DL, the WG should consider an appropriate status code, or set of status codes to be returned by a server during periods of resource unavailability. In my opinion, only a single status code would suffice, but we may want to consider a particular status code for each type of "resource" that is unavailable. There is a fairly likely case wherein transport layer resources are unavailable to complete a transport connection. In this event, the information available to clients as to what is actually causing resource exhaustion will be vague.

<u>Rationale for IPP-only Contention Handling</u>

I have purposely avoided addressing any type of specific contention handling of conditions that occur external to the "IPP domain". By outside the "IPP domain", I mean that I don't think we should attempt to "standardize" a contention solution that exposes any type of condition (potentially to the end user) that is "below" the IPP application protocol. This means specific HTTP or TCP/IP conditions that occur. Due to the differences in TCP/IP application programmer interfaces, and the differing types of HTTP interfaces that IPP might use (see environments 1 through 5 above), I think its difficult to come up with a concise, consistent behavior to document.

In my opinion, we should allow individual implementations to translate transport-level (HTTP, TCP/IP) conditions to the most informative indication to potential end-users. These transport-level contention conditions are few, and are not IPP-specific, and can occur in any applications that attempt to use the same transport mechanisms, so ambiguity is almost certain in some transport-level contention scenarios.

However, since the status codes and indications that we define for IPP are application-specific and connote more meaningful information related to the "printing" application, we should attempt to limit our scope to defining IPP-specific features for contention handling. We can also amortize this effort over all potential future mappings of IPP to other transports without having to re-address the problem.

Some of the potential "ratholes" that I would like to avoid are:

- Translation of NSAPI-based return codes to something usable by IPP
- Translation of ISAPI-based return codes to something usable by IPP
- Isolating System-V Transport Layer Interface (TLI) status codes for translation to something usable by IPP
- Translating BSD "errno" or Winsock "WSAGetLastError" status codes to something usable by IPP

- Any Java servlet-dependent status conditions
- Generic HTTP error code translation to something usable by IPP, especially in the presence of one or more intermediate proxies.