INTERNET-DRAFT Randy Turner
Expires: September 1998 Sharp Labs of America

Internet Printing Protocol over TCP/IP

Status of this Memo

1 Abstract

The Internet Printing Protocol (IPP) is fundamentally defined by the IPP Model & Semantics/1.0 Document [1]. The IPP model was designed to be transport-independent. There currently exists a document that describes using Hypertext Transfer Protocol (HTTP) 1.1 as a transport layer for IPP [IPPPROT]. Because the IPP model document is not transport-specific, it was envisioned that possibly multiple transport specifications would be authored for IPP. This document specifies such an alternate transport for IPP messages, and attempts to clarify the transport-independence implied by the IPP model and semantics document.

2 Overview

This document describes a new transport mapping for the IPP protocol. The existing set of documents describing IPP define a model and abstract protocol for printing, and an explicit encoding and transport over HTTP 1.1. This document is a transport document that explicit defines how the existing IPP encoding is transported directly over TCP.

This document makes explicit references to both the IPP model document [IPPMOD] and the existing IPP Protocol/Encoding document [IPPPROT]. This proposal implies no semantic changes to the IPP model document. Further, it reuses the encoding specified in [IPPPROT] in its entirety. Only the mechanism for transporting the existing encoding is modified by this proposal.

3 IPP over TCP/IP – Rationale Statement

There is a perceived notion that the current IPP-over-HTTP specification imposes a "heavyweight" requirement on low-cost, embedded devices, in terms of resources and implementation effort. Initial implementations of IPP-over-HTTP will be targeted towards server-based systems, with local storage capacity for spooling and other job management features. The use

of HTTP as a transport will allow quick deployment of internet capability for printing through standard HTTP server extension mechanisms (CGI, NSAPI, ISAPI, Java servlets, etc.). Because the core IPP protocol model contains no HTTP-specific requirements or semantics, this document suggests an alternate transport for the IPP abstract protocol utilizing simpler transport semantics, as well as providing slight changes to IPP client/server interaction. The changes are minor and allow tighter integration of client and printer for notifications and status information.

The following diagram shows one IPP topology for which the proposed TCP/IP transport would be utilized

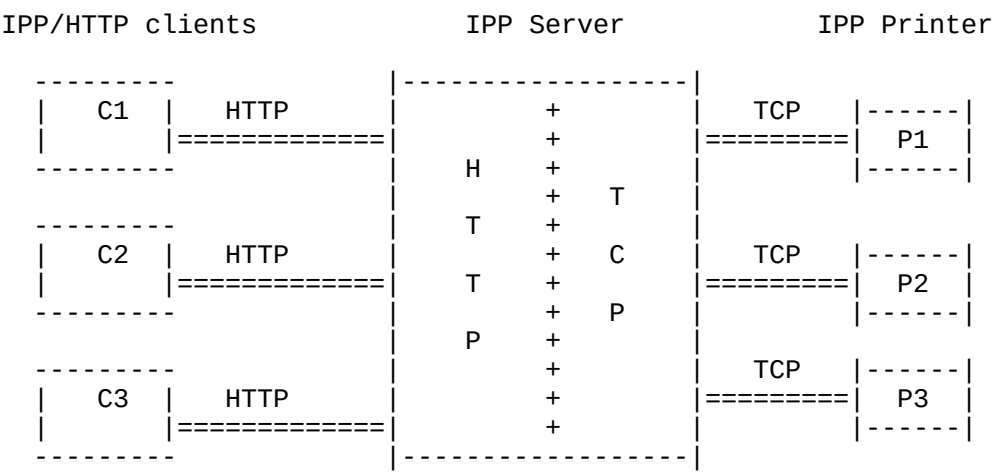

Figure 1

Existing IPP/1.0-over-HTTP clients would submit jobs to IPP servers. The servers would relay the IPP requests to physical printers using the proposed TCP/IP transport. The transport gateway is just that, a transport gateway, not an application-level gateway. Therefore, there would be no loss in application information from client to eventual physical printer.

Of course, this use of the proposed simple transport is but one possible topology. The diagram above could be changed so that both IPP servers and IPP clients BOTH connect to IPP physical printers, if both servers and clients wish to take advantage of the features of the proposed transport. Also, scenarios wherein multiple levels of servers communicating with servers which eventually communicate with a physical printer could be supported as well.

IPP security is also addressed by this memo, and a simpler mechanism for support of in-band security negotiation is included in the proposed transport.

4 IPP Protocol Processing

This draft proposes a model for IPP protocol operation that follows other application protocols that support multiple transports, including SNMP Version 3 [RFC2273]. The operation model described herein specifies two independently operating "layers": The IPP processing layer and one or more transport layers.

The IPP processing layer is the core protocol engine that understands the semantics of protocol operations, such as requests and responses. The core IPP protocol engine operates independently of transport. The independence is achieved through adherence to specific interfaces. The next section describes the abstract interface(s) employed to achieve the multiple-transport model. The discussion of abstract transport interfaces and subsequent status codes is merely to emphasize and clarify how a particular IPP implementation might be architected to support multiple transports. It also illustrates how IPP "transport-gateways" can be constructed. The inclusion of this abstract model does not imply that a particular implementation of this protocol mapping SHOULD or MUST be constructed using these abstract semantics.

4.1 IPP Transport Interface

The following abstract interface is used by an IPP processing engine to transmit a PDU, across a particular transport, to another IPP protocol processing engine. The model and format is taken from [RFC2273] as an example abstract interface, with similar features:

```
pduHandle = sendPdu(
         IN   transportDomain           -- transport domain to be used
         IN   transportAddress          -- destination network address
         IN   messageProcessingModel    -- IPP Version Number
         IN   securityModel             -- Security Model to use
         IN   securityName              -- on behalf of this principal
         IN   securityLevel             -- Level of Security requested
         IN   pduVersion                -- Encoding model used
         IN   PDU                       -- IPP Protocol Data Unit
         IN   expectResponse            -- TRUE or FALSE
              )
```

Where,

"transportDomain" is the particular transport over which the IPP PDU is being delivered.

"transportAddress" is the particular address within the transportDomain that should receive the PDU.

"messageProcessingModel" is the particular IPP version number for which the processing and semantics of the PDU are to be applied. Since the IPP core processing engine and the transport layer may be independently implemented, there might be version conflicts wherein a particular transport layer cannot support a particular version of the IPP model.

"securityModel" is the particular security mechanism being employed for protecting the PDU.

"securityLevel" is the particular level or degree of security within the "securityModel" used to convey this PDU.

"securityName" is a particular end-user identifier (if known) that should be used during the generation of authentication information for a particular security mechanism.

"pduVersion" is the particular encoding rules used to encode the PDU. This parameter is passed to the transport layer because the particular

transport layer might not be able to reliable encapsulate certain encodings and guarantee their delivery.

"sendPduHandle" is an opaque "transaction-id" generated by the specific transport layer for this PDU.

The core IPP processing engine would formulate IPP messages via some encoding, and then subsequently pass these encoded PDUs to the appropriate transport layer identified by transportDomain and endpoint identified by transportAddress. In actual client implementations, these two parameters would be derived from the "scheme" and "host" parts of a URI.

```
 pduLength = receivePdu(              -- process Response PDU
         IN   transportDomain
         IN   transportAddress
         IN   messageProcessingModel    -- IPP version number
         IN   securityModel             -- Security Model in use
         IN   securityLevel             -- Level of security
         IN   securityName              -- on behalf of this principal
         IN   pduVersion                -- encoding method used
         IN   PDU                       -- IPP Protocol Data Unit
         IN   sendPduHandle             -- handle from sendPDU
              )
```

Where,

"pduLength" is the length, in octets, of the received PDU. If the

"transportDomain" is the particular transport over which the IPP PDU is received.

"transportAddress" is the particular address within the transportDomain that originated the received PDU.

"messageProcessingModel" is the particular IPP version number for which the processing and semantics of the PDU are to be applied.

"securityModel" is the particular security mechanism being employed for protecting the received PDU.

"securityLevel" is the particular level or degree of security within the "securityModel" used to convey this PDU.

"securityName" is a particular end-user identifier (if known) that was translated from the particular security mechanism.

"pduVersion" is the particular encoding rules used to encode the received PDU.

"sendPduHandle" is an opaque "transaction-id" generated by the originator of this PDU.

This proposal specifies the use of the encoding as specified in [IPP-PROT]. One particular use of this transport specification would be to implement a gateway function as illustrated in Figure 1.

In the ideal gateway scenario, a core IPP processing engine would simply relay requests from one transport (HTTP) to another transport (TCP/IP), only varying the transportDomain and transportAddress parameters as necessary.

The primary motivation by creating these abstract interfaces is to allow maximum reuse of transport, encoding logic, and core protocol processing. Using this gateway scenario, no loss of IPP semantic information is incurred from end-user to IPP printer endpoint. In theory, it might even be possible to construct gateways using this model that are immune to differing IPP version numbers from client endpoint to printer endpoint. This is because in this example, no modification of the IPP PDU itself is performed when relaying from transport to transport. This assumption would of course have to undergo validation.

4.2 Transport-specific Header

This proposal allows some transport-specific capabilities not explicitly allowed (or guaranteed) by [IPPPROT]. This transport specification utilizes a IPP-specific transport header that is required to support mandatory features discussed in [IPPMOD]. This transport header can also provide capabilities not explicitly provided by [IPPMOD].

The transport header includes four fields, A pduLength field and a pduStatus field. The pduLength field denotes the length, in octets, of the PDU. The pduStatus field indicates the status of the particular PDU being processed. A list of possible status codes is discussed in section 5 of this proposal. The header itself is composed of ASCII text with the syntax described by the following ABNF:

```
Xpt-Header = pduStatus SEP pduLength SEP pdu

pduStatus = 1*DIGIT

pduLength = 1*DIGIT

pdu = OCTET-STRING

SEP = 0x13x10

OCTET-STRING = *BYTE

BYTE = 0x00..0x255
```

This specification also requires registration of a new URI scheme, "IPP", that designates a particular default port number for connecting to IPP services using the transport specified by this document. Note that the registration only specifies a default port number. Appendix A of this document is the complete text of the URL registration. The proposed URL syntax includes a field for specifying some other TCP port number other than the default.

5. Transport Layer Status Codes

The following transport-specific error codes are grouped into three different categories of severity: Normal, Error, and Warning. These status codes would be associated with the abstract transport interface previously described.

Normal

SUCCESS - transport layer request completed successfully

Errors

ERR-PROTOCOL - malformed transport layer packet was received
ERR-TIMEOUT - timeout waiting for response
ERR-DISCON - session abnormally disconnected
ERR-BADVER - IPP version not supported
ERR-BADPDU - IPP protocol encoding not supported
ERR-WOULDBLOCK - Initiating this operation would cause an indefinite blocking state to occur.
ERR-INTERNAL - An internal transport error occurred.

Warnings

WARN-MORE-DATA - More data is available from the transport layer for this particular request. This is an indication that more than one individual transport layer packet was necessary to contain a particular IPP message.

6. Security Considerations

The proposed transport specifies the use of one or more of the following Simple Authentication and Security Layer (SASL) [RFC2222] profiles:

- STARTTLS [STARTTLS]
- ANONYMOUS [RFC2245]
- CRAM-MD5 [RFC2195]

SASL allows the publication of only one URI for service discovery mechanisms (directory services, DHCP, etc). The existing IPP-over-HTTP specification requires the use of different URI publications for secure and non-secure IPP services. SASL negotiation is performed "in-band", over a single connection, thereby eliminating the need for different URIs for different security mechanisms. Of the three profiles specified above, all but the STARTTLS profile is available for referencing as an RFC. Given that this memo (IPP over TCP/IP) is dated March 1998, the draft that describes the STARTTLS profile is scheduled for last call at the beginning of April 1998. Anticipated RFC status for this draft falls within a reasonable time period for inclusion in this proposal.

This document suggests the use of both ANONYMOUS and CRAM-MD5 as MANDATORY security mechanisms. Both of these mechanisms only provide authentication, not privacy. If privacy is required, then, like the IPP model document specifies, TLS should be negotiated using the STARTTLS SASL profile.

The ANONYMOUS mechanism allows similar access semantics as "anonymous FTP", using just a simple clear text id string such as an email address or other simple ASCII string.

CRAM-MD5 is a very simple mechanism for authentication using information that is not passed as clear text. CRAM-MD5 like other MD5-based authentication schemes, requires the knowledge of a shared secret between client and printer. Shared secrets do not need to be kept for all possible end-users. Rather, administrators may want to provide secret keys that map to either small groups, or departmental access keys. However, the use of aggregated keys does allow a greater possibility for keys to be revealed to non-authorized parties. On the

other hand, this type of security may be acceptable for certain environments that do not want open access to services(ANONYMOUS), but do not want to deal with TLS-based security. [RFC2104] contains a detailed description of the keyed-MD5 method employed by CRAM-MD5, as well as example code that implements the algorithm.

6. References

[IPPMOD] DeBry R., Hastings T., Herriot R., Isaacson S., Powell P., "Internet Printing Protocol/1.0: Model and Semantics", Internet-Draft draft-ietf-ipp-model-09, January 1998

[IPPPROT] Herriot R., Butler S., Moore P., Turner R., "Internet Printing Protocol/1.0: Protocol Specification", Internet-Draft draft-ietf-ipp-protocol-05, January 1998

[RFC1759] Smith R., Wright F., Hastings T., Zilles S., Gyllenskog J., "Printer MIB", RFC 1759, March 1995

[RFC2273] Levi D., Meyer P., Stewart B., "SNMPv3 Applications", RFC 2273, January 1998

[RFC2068] Fielding R., Gettys J., Mogul J., Frystyk H., Berners-Lee T., "Hypertext Transfer Protocol – HTTP/1.1", RFC 2068, January 1997

[RFC2119] Bradner S., "Keywords for Use in RFCs to Indicate Requirement Levels", RFC 2219, March 1997

[RFC2222] Myers J., "Simple Authentication and Security Layer (SASL)", RFC 2222, October 1997

[RFC2245] Newman C., "Anonymous SASL Mechanism", RFC 2245, November 1997

[RFC-2104] Krayczyk H., Bellare M., Canetti R., "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997

[RFC-2195] Klensin J., Catoe R., Krumviede P., "IMAP/POP AUTHorize Extension for Simple Challenge/Response", RFC 2195, September 1997

[STARTTLS] Newman C., "Using TLS with IMAP4, POP3, and ACAP", Internet-Draft draft-newman-tls-imappop-03, March 1998

Appendix A – "IPP" Scheme Registration

The following IPP scheme proposal is meant to start debate on IPP schemed URLs. The scheme suggested below would be advertised by servers to potential clients.

IPP://host.domain[:port]

Clients attempting access to a resource identified by the "IPP" scheme MUST utilize the IPP transport mapping specified by this document.