Network Working Group
Request for Comments: 3252
Category: Informational

H. Kennedy
Mimezine

1 April 2002

Binary Lexical Octet Ad-hoc Transport

Status of this Memo

Copyright Notice

Abstract

This document defines a reformulation of IP and two transport layer protocols (TCP and UDP) as XML applications.

1. Introduction

1.1. Overview

This document describes the Binary Lexical Octet Ad-hoc Transport (BLOAT): a reformulation of a widely-deployed network-layer protocol (IP [RFC791]), and two associated transport layer protocols (TCP [RFC793] and UDP [RFC768]) as XML [XML] applications. It also describes methods for transporting BLOAT over Ethernet and IEEE 802 networks as well as encapsulating BLOAT in IP for gatewaying BLOAT across the public Internet.

1.2. Motivation

The wild popularity of XML as a basis for application-level protocols such as the Blocks Extensible Exchange Protocol [RFC3080], the Simple Object Access Protocol [SOAP], and Jabber [JABBER] prompted investigation into the possibility of extending the use of XML in the protocol stack. Using XML at both the transport and network layer in addition to the application layer would provide for an amazing amount of power and flexibility while removing dependencies on proprietary and hard-to-understand binary protocols. This protocol unification would also allow applications to use a single XML parser for all aspects of their operation, eliminating developer time spent figuring out the intricacies of each new protocol, and moving the hard work of

parsing to the XML toolset. The use of XML also mitigates concerns over "network vs. host" byte ordering which is at the root of many network application bugs.

1.3. Relation to Existing Protocols

The reformulations specified in this RFC follow as closely as possible the spirit of the RFCs on which they are based, and so MAY contain elements or attributes that would not be needed in a pure reworking (e.g. length attributes, which are implicit in XML.)

The layering of network and transport protocols are maintained in this RFC despite the optimizations that could be made if the line were somewhat blurred (i.e. merging TCP and IP into a single, larger element in the DTD) in order to foster future use of this protocol as a basis for reformulating other protocols (such as ICMP.)

Other than the encoding, the behavioral aspects of each of the existing protocols remain unchanged. Routing, address spaces, TCP congestion control, etc. behave as specified in the extant standards. Adapting to new standards and experimental algorithm heuristics for improving performance will become much easier once the move to BLOAT has been completed.

1.4. Requirement Levels

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in BCP 14, RFC 2119 [RFC2119].

2. IPoXML

This protocol MUST be implemented to be compliant with this RFC. IPoXML is the root protocol REQUIRED for effective use of TCPoXML (section 3.) and higher-level application protocols.

The DTD for this document type can be found in section 7.1.

The routing of IPoXML can be easily implemented on hosts with an XML parser, as the regular structure lends itself handily to parsing and validation of the document/datagram and then processing the destination address, TTL, and checksum before sending it on to its next-hop.

The reformulation of IPv4 was chosen over IPv6 [RFC2460] due to the wider deployment of IPv4 and the fact that implementing IPv6 as XML would have exceeded the 1500 byte Ethernet MTU.

All BLOAT implementations MUST use - and specify - the UTF-8 encoding of RFC 2279 [RFC2279]. All BLOAT document/datagrams MUST be well-formed and include the XMLDecl.

2.1. IP Description

A number of items have changed (for the better) from the original IP specification. Bit-masks, where present have been converted into human-readable values. IP addresses are listed in their dotted-decimal notation [RFC1123]. Length and checksum values are present as decimal integers.

To calculate the length and checksum fields of the IP element, a canonicalized form of the element MUST be used. The canonical form SHALL have no whitespace (including newline characters) between elements and only one space character between attributes. There SHALL NOT be a space following the last attribute in an element.

An iterative method SHOULD be used to calculate checksums, as the length field will vary based on the size of the checksum.

The payload element bears special attention. Due to the character set restrictions of XML, the payload of IP datagrams (which MAY contain arbitrary data) MUST be encoded for transport. This RFC REQUIRES the contents of the payload to be encoded in the base-64 encoding of RFC 2045 [RFC2045], but removes the requirement that the encoded output MUST be wrapped on 76-character lines.

2.2. Example Datagram

The following is an example IPoXML datagram with an empty payload:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE ip PUBLIC "-//IETF//DTD BLOAT 1.0 IP//EN" "bloat.dtd">
<ip>
<header length="474">
<version value="4"/>
<tos precedence="Routine" delay="Normal" throughput="Normal"
     relibility="Normal" reserved="0"/>
<total.length value="461"/>
<id value="1"/>
<flags reserved="0" df="dont" mf="last"/>
<offset value="0"/>
<ttl value="255"/>
<protocol value="6"/>
<checksum value="8707"/>
<source address="10.0.0.22"/>
<destination address="10.0.0.1"/>
<options>
<end copied="0" class="0" number="0"/>
</options>
<padding pad="0"/>
</header>
<payload>
</payload>
</ip>
```

3. TCPoXML

This protocol MUST be implemented to be compliant with this RFC. The DTD for this document type can be found in section 7.2.

3.1. TCP Description

A number of items have changed from the original TCP specification. Bit-masks, where present have been converted into human-readable values. Length and checksum and port values are present as decimal integers.

To calculate the length and checksum fields of the TCP element, a canonicalized form of the element MUST be used as in section 2.1.

An iterative method SHOULD be used to calculate checksums as in section 2.1.

The payload element MUST be encoded as in section 2.1.

The TCP offset element was expanded to a maximum of 255 from 16 to allow for the increased size of the header in XML.

TCPoXML datagrams encapsulated by IPoXML MAY omit the <?xml?> header as well as the <!DOCTYPE> declaration.

3.2. Example Datagram

The following is an example TCPoXML datagram with an empty payload:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE tcp PUBLIC "-//IETF//DTD BLOAT 1.0 TCP//EN" "bloat.dtd">
<tcp>
<tcp.header>
<src port="31415"/>
<dest port="42424"/>
<sequence number="322622954"/>
<acknowledgement number="689715995"/>
<offset number=""/>
<reserved value="0"/>
<control syn="1" ack="1"/>
<window size="1"/>
<urgent pointer="0"/>
<checksum value="2988"/>
<tcp.options>
<tcp.end kind="0"/>
</tcp.options>
<padding pad="0"/>
</tcp.header>
<payload>
</payload>
</tcp>
```

4. UDPoXML

This protocol MUST be implemented to be compliant with this RFC. The DTD for this document type can be found in section 7.3.

4.1. UDP Description

A number of items have changed from the original UDP specification. Bit-masks, where present have been converted into human-readable values. Length and checksum and port values are present as decimal integers.

To calculate the length and checksum fields of the UDP element, a canonicalized form of the element MUST be used as in section 2.1. An iterative method SHOULD be used to calculate checksums as in section 2.1.

The payload element MUST be encoded as in section 2.1.

UDPoXML datagrams encapsulated by IPoXML MAY omit the <?xml?> header as well as the <!DOCTYPE> declaration.

4.2. Example Datagram

The following is an example UDPoXML datagram with an empty payload:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE udp PUBLIC "-//IETF//DTD BLOAT 1.0 UDP//EN" "bloat.dtd">
<udp>
<udp.header>
<src port="31415"/>
<dest port="42424"/>
<udp.length value="143"/>
<checksum value="2988"/>
</udp.header>
<payload>
</payload>
</udp>
```

5. Network Transport

This document provides for the transmission of BLOAT datagrams over two common families of physical layer transport. Future RFCs will address additional transports as routing vendors catch up to the specification, and we begin to see BLOAT routed across the Internet backbone.

5.1. Ethernet

BLOAT is encapsulated in Ethernet datagrams as in [RFC894] with the exception that the type field of the Ethernet frame MUST contain the value 0xBEEF. The first 5 octets of the Ethernet frame payload will be 0x3c 3f 78 6d 6c ("<?xml".)

5.2. IEEE 802

BLOAT is encapsulated in IEEE 802 Networks as in [RFC1042] except that the protocol type code for IPoXML is 0xBEEF.

6. Gatewaying over IP

In order to facilitate the gradual introduction of BLOAT into the public Internet, BLOAT MAY be encapsulated in IP as in [RFC2003] to gateway between networks that run BLOAT natively on their LANs.

7. DTDs

The Transport DTDs (7.2. and 7.3.) build on the definitions in the Network DTD (7.1.)

The DTDs are referenced by their PubidLiteral and SystemLiteral (from [XML]) although it is understood that most IPoXML implementations will not need to pull down the DTD, as it will normally be embedded in the implementation, and presents something of a catch-22 if you need to load part of your network protocol over the network.

7.1. IPoXML DTD

```
<!--
 DTD for IP over XML.
 Refer to this DTD as:

 <!DOCTYPE ip PUBLIC "-//IETF//DTD BLOAT 1.0 IP//EN" "bloat.dtd">
-->
<!--
 DTD data types:

   Digits      [0..9]+

   Precedence  "NetworkControl | InternetworkControl |
                CRITIC | FlashOverride | Flash | Immediate |
                Priority | Routine"

   IP4Addr     "dotted-decimal" notation of [RFC1123]

   Class       [0..3]

   Sec          "Unclassified | Confidential | EFTO | MMMM | PROG |
                 Restricted | Secret | Top Secret | Reserved"

   Compartments [0..65535]

   Handling     [0..65535]

   TCC          [0..16777216]

-->
```

```
<!ENTITY % Digits "CDATA">
<!ENTITY % Precedence "CDATA">
<!ENTITY % IP4Addr "CDATA">
<!ENTITY % Class "CDATA">
<!ENTITY % Sec "CDATA">
<!ENTITY % Compartments "CDATA">
<!ENTITY % Handling "CDATA">
<!ENTITY % TCC "CDATA">

<!ELEMENT ip (header, payload)>

<!ELEMENT header (version, tos, total.length, id, flags, offset, ttl,
                  protocol, checksum, source, destination, options,
                  padding)>
<!-- length of header in 32-bit words -->
<!ATTLIST header
          length %Digits; #REQUIRED>

<!ELEMENT version EMPTY>
<!-- ip version. SHOULD be "4" -->
<!ATTLIST version
          value   %Digits;  #REQUIRED>

<!ELEMENT tos EMPTY>
<!ATTLIST tos
          precedence   %Precedence;    #REQUIRED
          delay    (normal | low)  #REQUIRED
          throughput   (normal | high) #REQUIRED
          relibility   (normal | high) #REQUIRED
          reserved     CDATA #FIXED "0">

<!ELEMENT total.length EMPTY>
<!--
 total length of datagram (header and payload) in octets, MUST be
 less than 65,535 (and SHOULD be less than 1024 for IPoXML on local
 ethernets).
-->
<!ATTLIST total.length
          value %Digits; #REQUIRED>

<!ELEMENT id EMPTY>
<!-- 0 <= id <= 65,535  -->
<!ATTLIST id
          value %Digits; #REQUIRED>

<!ELEMENT flags EMPTY>
<!-- df = don't fragment, mf = more fragments  -->
<!ATTLIST flags
```

```
          reserved CDATA  #FIXED "0"
          df (may|dont)   #REQUIRED
          mf (last|more)  #REQUIRED>

   <!ELEMENT offset EMPTY>
   <!-- 0 <= offset <= 8192 measured in 8 octet (64-bit) chunks -->
   <!ATTLIST offset
             value %Digits; #REQUIRED>

   <!ELEMENT ttl EMPTY>
   <!-- 0 <= ttl <= 255 -->
   <!ATTLIST ttl
             value %Digits; #REQUIRED>

   <!ELEMENT protocol EMPTY>
   <!-- 0 <= protocol <= 255 (per IANA) -->
   <!ATTLIST protocol
             value %Digits; #REQUIRED>

   <!ELEMENT checksum EMPTY>
   <!-- 0 <= checksum <= 65535 (over header only) -->
   <!ATTLIST checksum
             value %Digits; #REQUIRED>

   <!ELEMENT source EMPTY>
   <!ATTLIST source
             address %IP4Addr; #REQUIRED>

   <!ELEMENT destination EMPTY>
   <!ATTLIST destination
             address %IP4Addr; #REQUIRED>

   <!ELEMENT options ( end | noop | security | loose | strict | record
                     | stream | timestamp )*>

   <!ELEMENT end EMPTY>
   <!ATTLIST end
             copied (0|1) #REQUIRED
             class  CDATA #FIXED "0"
             number CDATA #FIXED "0">

   <!ELEMENT noop EMPTY>
   <!ATTLIST noop
             copied (0|1) #REQUIRED
             class  CDATA #FIXED "0"
             number CDATA #FIXED "1">

   <!ELEMENT security EMPTY>
```

```
   <!ATTLIST security
             copied CDATA #FIXED "1"
             class  CDATA #FIXED "0"
             number CDATA #FIXED "2"
             length CDATA #FIXED "11"
             security %Sec; #REQUIRED
             compartments %Compartments; #REQUIRED
             handling %Handling; #REQUIRED
             tcc %TCC; #REQUIRED>
   <!ELEMENT loose (hop)+>
   <!ATTLIST loose
             copied CDATA #FIXED "1"
             class  CDATA #FIXED "0"
             number CDATA #FIXED "3"
             length %Digits; #REQUIRED
             pointer %Digits; #REQUIRED>

   <!ELEMENT hop EMPTY>
   <!ATTLIST hop
             address %IP4Addr; #REQUIRED>

   <!ELEMENT strict (hop)+>
   <!ATTLIST strict
             copied CDATA #FIXED "1"
             class  CDATA #FIXED "0"
             number CDATA #FIXED "9"
             length %Digits; #REQUIRED
             pointer %Digits; #REQUIRED>

   <!ELEMENT record (hop)+>
   <!ATTLIST record
             copied CDATA #FIXED "0"
             class  CDATA #FIXED "0"
             number CDATA #FIXED "7"
             length %Digits; #REQUIRED
             pointer %Digits; #REQUIRED>

   <!ELEMENT stream EMPTY>
   <!-- 0 <= id <= 65,535 -->
   <!ATTLIST stream
             copied CDATA #FIXED "1"
             class  CDATA #FIXED "0"
             number CDATA #FIXED "8"
             length CDATA #FIXED "4"
             id %Digits; #REQUIRED>

   <!ELEMENT timestamp (tstamp)+>
   <!-- 0 <= oflw <=15 -->
```

```
   <!ATTLIST timestamp
             copied CDATA #FIXED "0"
             class  CDATA #FIXED "2"
             number CDATA #FIXED "4"
             length %Digits;  #REQUIRED
             pointer %Digits; #REQUIRED
             oflw %Digits;    #REQUIRED
             flag (0 | 1 | 3)  #REQUIRED>

   <!ELEMENT tstamp EMPTY>
   <!ATTLIST tstamp
             time %Digits;   #REQUIRED
             address %IP4Addr; #IMPLIED>
   <!--
       padding to bring header to 32-bit boundary.
       pad MUST be "0"*
    -->
   <!ELEMENT padding EMPTY>
   <!ATTLIST padding
             pad CDATA #REQUIRED>

   <!-- payload MUST be encoded as base-64 [RFC2045], as modified
        by section 2.1 of this RFC -->
   <!ELEMENT payload (CDATA)>
```

7.2. TCPoXML DTD

```
   <!--
      DTD for TCP over XML.
      Refer to this DTD as:

      <!DOCTYPE tcp PUBLIC "-//IETF//DTD BLOAT 1.0 TCP//EN" "bloat.dtd">
   -->

   <!-- the pseudoheader is only included for checksum calculations -->
   <!ELEMENT tcp (tcp.pseudoheader?, tcp.header, payload)>

   <!ELEMENT tcp.header (src, dest, sequence, acknowledgement, offset,
                         reserved, control, window, checksum, urgent,
                         tcp.options, padding)>

   <!ELEMENT src EMPTY>
   <!-- 0 <= port <= 65,535 -->
   <!ATTLIST src
             port %Digits; #REQUIRED>

   <!ELEMENT dest EMPTY>
   <!-- 0 <= port <= 65,535 -->
```

```
   <!ATTLIST dest
             port %Digits; #REQUIRED>

   <!ELEMENT sequence EMPTY>
   <!-- 0 <= number <= 4294967295 -->
   <!ATTLIST sequence
             number %Digits; #REQUIRED>

   <!ELEMENT acknowledgement EMPTY>
   <!-- 0 <= number <= 4294967295 -->
   <!ATTLIST acknowledgement
             number %Digits; #REQUIRED>

   <!ELEMENT offset EMPTY>
   <!-- 0 <= number <= 255 -->
   <!ATTLIST offset
             number %Digits; #REQUIRED>

   <!ELEMENT reserved EMPTY>
   <!ATTLIST reserved
             value CDATA #FIXED "0">

   <!ELEMENT control EMPTY>
   <!ATTLIST control
             urg (0|1) #IMPLIED
             ack (0|1) #IMPLIED
             psh (0|1) #IMPLIED
             rst (0|1) #IMPLIED
             syn (0|1) #IMPLIED
             fin (0|1) #IMPLIED>

   <!ELEMENT window EMPTY>
   <!-- 0 <= size <= 65,535 -->
   <!ATTLIST window
             size %Digits; #REQUIRED>

   <!--
      checksum as in ip, but with
      the following pseudo-header added into the tcp element:
     -->
   <!ELEMENT tcp.pseudoheader (source, destination, protocol,
                               tcp.length)>

   <!--
      tcp header + data length in octets. does not include the size of

      the pseudoheader.
    -->
```

```
   <!ELEMENT tcp.length EMPTY>
   <!ATTLIST tcp.length
             value %Digits; #REQUIRED>

   <!ELEMENT urgent EMPTY>
   <!-- 0 <= pointer <= 65,535 -->
   <!ATTLIST urgent
             pointer %Digits; #REQUIRED>

   <!ELEMENT tcp.options (tcp.end | tcp.noop | tcp.mss)+>

   <!ELEMENT tcp.end EMPTY>
   <!ATTLIST tcp.end
             kind CDATA #FIXED "0">

   <!ELEMENT tcp.noop EMPTY>
   <!ATTLIST tcp.noop
             kind CDATA #FIXED "1">

   <!ELEMENT tcp.mss EMPTY>
   <!ATTLIST tcp.mss
             kind CDATA #FIXED "2"
             length CDATA #FIXED "4"
             size %Digits; #REQUIRED>
```

7.3. UDPoXML DTD

```
   <!--
      DTD for UDP over XML.
      Refer to this DTD as:

      <!DOCTYPE udp PUBLIC "-//IETF//DTD BLOAT 1.0 UDP//EN" "bloat.dtd">
   -->

   <!ELEMENT udp (udp.pseudoheader?, udp.header, payload)>

   <!ELEMENT udp.header (src, dest, udp.length, checksum)>

   <!ELEMENT udp.pseudoheader (source, destination, protocol,
                               udp.length)>

   <!--
      udp header + data length in octets. does not include the size of
      the pseudoheader.
    -->
   <!ELEMENT udp.length EMPTY>
   <!ATTLIST udp.length
             value %Digits; #REQUIRED>
```

8. Security Considerations

XML, as a subset of SGML, has the same security considerations as specified in SGML Media Types [RFC1874]. Security considerations that apply to IP, TCP and UDP also likely apply to BLOAT as it does not attempt to correct for issues not related to message format.

9. References

[JABBER] Miller, J., "Jabber", draft-miller-jabber-00.txt, February 2002. (Work in Progress)

[RFC768] Postel, J., "User Datagram Protocol", STD 6, RFC 768, August 1980.

[RFC791] Postel, J., "Internet Protocol", STD 5, RFC 791, September 1981.

[RFC793] Postel, J., "Transmission Control Protocol", STD 7, RFC 793, September 1981.

[RFC894] Hornig, C., "Standard for the Transmission of IP Datagrams over Ethernet Networks.", RFC 894, April 1984.

[RFC1042] Postel, J. and J. Reynolds, "Standard for the Transmission of IP Datagrams Over IEEE 802 Networks", STD 43, RFC 1042, February 1988.

[RFC1123] Braden, R., "Requirements for Internet Hosts - Application and Support", RFC 1123, October 1989.

[RFC1874] Levinson, E., "SGML Media Types", RFC 1874, December 1995.

[RFC2003] Perkins, C., "IP Encapsulation within IP", RFC 2003, October 1996.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, March 1997.

[RFC2279] Yergeau, F., "UTF-8, a transformation format of ISO 10646", RFC 2279, January 1998.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

[RFC3080] Rose, M., "The Blocks Extensible Exchange Protocol Core", RFC 3080, March 2001.

[SOAP] Box, D., Ehnebuske, D., Kakivaya, G., Layman, A., Mendelsohn, N., Nielsen, H. F., Thatte, S. Winer, D., "Simple Object Access Protocol (SOAP) 1.1" World Wide Web Consortium Note, May 2000 http://www.w3.org/TR/SOAP/

[XML] Bray, T., Paoli, J., Sperberg-McQueen, C. M., "Extensible Markup Language (XML)" World Wide Web Consortium Recommendation REC- xml-19980210. http://www.w3.org/TR/1998/REC-xml-19980210

10. Author's Address

Hugh Kennedy
Mimezine
1060 West Addison
Chicago, IL 60613
USA

EMail: kennedyh@engin.umich.edu

11. Full Copyright Statement

Acknowledgement

Funding for the RFC Editor function is currently provided by the Internet Society.