DTN Research Group
Internet-Draft
Intended status: Experimental
Expires: May 15, 2008

J. Ott
T. Kaerkkaeinen
TKK Netlab
M. Pitkaenen
Helsinki Institute of Physics
(HIP) Technology Programme
November 12, 2007

Application Conventions for Bundle-based Communications
draft-ott-dtnrg-dtn-appl-00.txt

Status of this Memo

Copyright Notice

Abstract

This document discusses issues arising when moving the bundle protocol usage from closed or lab deployments into open, heterogeneous environments with a variety of usages and applications carried over common bundle agents. We address naming conventions and endpoint identifiers, content encapsulation, and on the identification of application contents.

Table of Contents

1. Introduction

Communications using the bundle protocol specification [2] has so far mostly been carried out in rather closed environments. That is, bundle agents have typically been handling messages from either a single type of application or from multiple applications developed in a coordinated fashion. Common usage conventions for the URIs for endpoint identification and bundle routing are not an issue in such environments. Recent research, however, has suggested the use of Delay-tolerant Networking based upon the DTNRG architecture [6] for a number of open applications to be run across the Internet or connected to the Internet, suggesting the use of DTN for addressing basic communication needs in remote areas and in support of mobility. In this context, various application layer protocols from the Internet realm have been adapted for transmission over the bundle protocol, most notably email and HTTP, with further protocols under investigation.

Building upon some of the thoughts put forward in [12], this document identifies some further issues requiring consideration when moving from the aforementioned "closed" DTN operation to an open environment and suggests usage conventions for the URIs for endpoint identification and bundle routing that allow arbitrary applications to co-exist in a shared DTN environment. In addition, this document proposes an---initially experimental---extension to the bundle protocol header in support of application-specific functions.

This document is based upon experience gained from using and extending the DTN reference implementation, expanding on it, and writing prototype applications and protocol encapsulations. Our experience is based upon supporting delay-tolerant interpersonal communications; other application scenarios will bring along further and possibly different requirements and observations. Comments and additions are solicited to create a more comprehensive view of the problems at hand when moving from closed to open deployments of DTNs.

For draft specifications in this document, the keywords MUST, SHOULD, and MAY as defined in RFC 2119 [1] apply.

2. DTN Operation in an Open Environment

When running DTN applications in the open Internet, and ultimately at large scale, bundle routers form a DTN overlay of some sort (which may go well beyond the 'borders' of the Internet of today). If arbitrary applications are run in such an environment, it is necessary that (see also [12]):

- different types of applications choose their endpoint identifiers (EIDs) from disjoint namespaces so that different application instances do not accidentally collide with one another;
- different applications of the same type choose their EIDs from the same namespace but still ensure that these are unique and allow for adequate multiplexing within the application class where necessary;
- bundle agents cooperating to serve multiple applications agree on a set of rules for routing based upon EIDs so that bundles are forwarded to their respective destinations; and
- the usage of EIDs across the DTN allows for efficient routing protocols if dynamic exchange of routing information is deemed necessary (how to exchange the routing information is yet another aspect).

This calls for common usage conventions for the URIs for endpoint identification and bundle routing. The basic URI conventions are addressed in section 3, discussions on routing will be captured in section 4 (to be expanded further in a future revision of this document).

Furthermore, per-application conventions should be provided that specify how application protocol messages are encapsulated in bundles. In general, this may include

- message encoding rules,
- how to delineate messages (if stacking multiple messages is allowed in the same bundle),
- specifying the message and content type,
- possibly specifying rules for (reactive) fragmentation,
- the utilization of bundle-layer mechanisms (such as end-to-end delivery confirmations), and

- possibly transfer-encodings and other operations (such as compression of the message contents).

To a certain extent, guidelines---somewhat similar to those specified for writers of RTP payload types [8] [13]---are conceivable. Such a set of encapsulation rules for text-based Internet application protocols is addressed in section 5.

Finally, bundle routing and forwarding schemes may benefit from understanding, at an abstract level, the application and message type of a particular message. Depending on the URI conventions used, some of this information may already be extracted from the EID. In order to decouple endpoint addressing from application-independent bundle contents identification, we suggest an extension block for the bundle protocol that captures common information about the bundle contents. This is specified in section 6.

3. Conventions for DTN EIDs

Endpoint identifiers (EIDs) are used to denote individual endpoints (singletons) or groups of endpoints. They serve as the primary point for routing/forwarding towards a (set of) physical/logical node(s) as well as for demultiplexing different applications running on a single node. EIDs are defined to be Unique Resource Identifiers (URIs) and thus comprise a 'scheme' and a 'scheme-specific part' (SSP), all text-encoded and separated by the colon (':') character [2].

Some considerations on EIDs have been raised in [12] which explains a lot of the open issues with EIDs. This document expands on these with some concrete proposals.

Currently, one URI scheme for DTNs ('dtn:') is suggested with no explicit substructure defined for the SSP. Only 'dtn:none' is defined [2]. However, the DTN reference implementation uses a structure of 'dtn:<node-id>/<application-id>' which has been described in earlier versions of the bundle protocol specification.

To perform demultiplexing across different applications, the respective application-specific URI schemes MUST be used, e.g.,

- 'mailto:' to identify senders and receivers of email messages encapsulated in bundles
- 'http:' to identify HTTP traffic
- 'ftp:' to denote file transfer operations

This is in contrast to an earlier proposal contained in the bundle specification about using suffixes of the EID to denote individual applications (which is still, however, one of the practices supported in the DTN reference implementation).

The demultiplexing beyond the application protocol is not defined in this document and is considered to be application-specific. However, note that some guidelines are necessary which deserve common discussion:

- For many applications, well-known addresses used to contact a server or peer instance on a particular node (the "listening EID") may need to be distinguished from application instances initiating outgoing communications (the latter of which may be assigned "dynamic EIDs") so that multiple instances of an application can be instantiated on a node. One simple approach would be to use a base EID for the well-known address and append application-specific instance numbers (as has been done in the DTN reference

implementation and for various applications). For applications that do not perform a specific registrations when initiating communications to a peer randomly assigned identifiers could be used.

- Whenever one DTN endpoint acts as a representative for multiple others, discussion is needed if and how a differentiation between the final target and its representative shall take place. One example are email message stores (servers) which maintain messages for a set of users: for (interactive) access and/or for backup purposes. Another one is the case of communication between DTN realms and applications in the Internet, e.g., via DTN-to-Internet gateways as have been designed for email.

Another interesting aspect is how much application context an EID should include. Two basic options are conceivable:

- The EID includes only sufficient information to identify the application entity itself while application-specific information is not part of the EID. For an HTTP URI, this would mean that the HTTP URI does not contain a local resource (but simply http://hostname/). It needs to be verified that such reductions are allowed in the respective URI formats.

- The EID includes additional information about the local resource and the respective application entity is found by means of (local) prefix or suffix matching. For an HTTP URI, this would mean that the complete resource URI is included (http://hostname/path). Particularly in this case, wildcard matching for routing and forwarding as well as for registrations is required to allow "server applications" to easily take responsibility for a large set of resources.

Combinations are also possible. On the one hand, allowing further demultiplexing information in the EID may simplify application-specific processing as no further intra-application demultiplexing points are needed and load balancing across multiple instances of the same application may be simplified. On the other hand, disallowing demultiplexing may simplify aggregation of routing information. However, it should be considered that, unlike the current Internet, DTNs may not use routing schemes that benefit from hierarchical structures and URIs (e.g., those belonging to users such as mailto: URIs) may not be singletons as they have been in the traditional Internet.

Further considerations on routing using different EID schemes are given in the next section.

4. Routing conventions and EIDs

At this point, this document restricts itself to outlining the options for DTN routing based upon EIDs to stimulate discussion. Future revisions will incorporate findinds from such discussions and give recommendations on how DTN routing protocols should be defined to make use of EIDs.

Basically, there are several alternatives for DTN routing protocols:

a. Routing protocols may treat EIDs as opaque strings without any substructure and perform forwarding decisions only upon full matches.

b. Routing protocols may treat EIDs as opaque strings without any substructure but assume significance to decrease from left to right. In such a case, they may perform forwarding decisions based upon longest prefix matches.

c. Routing protocols may utilize the URI structure of EIDs being composed of a 'scheme' and a 'scheme-specific part' (SSP). They may choose to route independent of the respective scheme and treat the SSP either as an opaque string and perform routing as per (a) or (b) on the SSP.

d. Like (c), however, routing may be dependent on the respective scheme. The forwarding decision may treat the scheme as more important than the SSP (and, e.g., choose the forwarding table based upon the scheme) or less important (and, e.g., take only scheduling or load balancing decisions based upont the scheme). In either case, the SSP may be treated as an opaque string and perform routing as per (a) or (b) on the SSP.

e. Like (c) and (d), however, the scheme-specific part is treated as having a substructure composed of one part (A) denoting a host or a topologically or administratively significant entity to be exploited for routing purposes and another part (B) identifying a sub-entity within the former. In the traditional Internet, this would be an IP address (A) and a port number (B). For HTTP URIs, this would be a "hostport" part identifying the server (A) and the identifier for the local resource (B). For mailto and other URIs identifying users, this would be the user identifier (B) and a host/domain name (A), with the sequence being reversed. In these cases, a hierarchical routing may be applied that first tries to route towards the (physical) entity (e.g., a host) using (c) or (d) on part (A) but does not consider part (B) for 'global' forwarding. Part (B) will only be considered for 'local' delivery.

f. Within part (A) identified in (e), some further substructure may be considerd, e.g., following a hierarchy similar to the DNS (but not necessarily resolvable via DNS). If some substructure is assumed, further hierarchical routing and route aggregation are possible. It should be noted that, with current practice in the DTN URI scheme, such a hierarchy may be present but does not map to the DNS and hence cannot be used to resolve DTN URIs into specific bundle routers. For specific URI schemes, such a mapping may be specified at some point; however, in general, no such mapping can be assumed.

g. Finally, some form of wildcard or full regular expression-based matching could be applied to the entire EID (and thus, implicitly, also separately to both parts (A) and (B)) to allow for utmost flexibility.

Obviously, the definition of URI conventions from section 3 and the routing/forwarding conventions specified in section 5 require some degree of coordination.

In particular, when allowing for arbitrary URI schemes, hierarchical routing may only be possible if the URI schemes are understood by all bundle agents. This requires careful definition of the routing conventions to maintain extensibility towards future URI schemes.

Two simple examples of different ordering of hierarchies:

mailto:jo@example.com

http://www.example.com/~jo/

Unless a well-known (at the time of the initial specification) scheme is used, an EID could explicitly indicate whether prefix matching should be done from left to right or from the right to the left and which parts should be considered. This is depicted in figure 1:

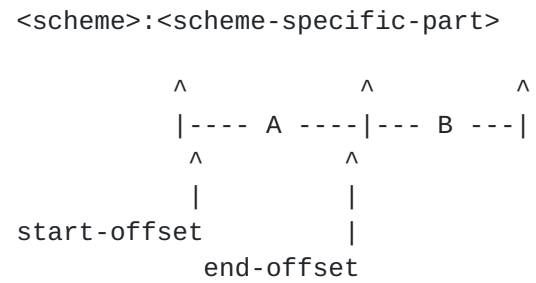


Figure 1: Indicating forwarding-relevant information in an EID

For this purpose, three additional fields (optional) would be required (operating on octets):

- start-offset for the 'routable part' of the EID>
- end-offset for the 'routable part' of the EID
- matching-flag: indicates 'FULL', 'LP', or 'LS' to denote the type of matching (full, longest-prefix, or longest-suffix')

Recent experience when using the DTN reference implementation for realizing an email gateway confirms the need for conventions to handle routing based upon EIDs. In the specific case, the DTN reference implementation interprets mailto URIs (mailto:user@domain) and disregards the part left of the '@' sign (i.e., the user) for routing. While this supports efficient routing towards a representative mail server (as discussed above), it prohibits user-specific routing based upon the full EID in ad-hoc networks. An explicit expression of how the respective EID is to be treated internally by intermediate nodes hence is advisable. Proper hints should ideally allow for both delivery to individual users as well as delivery via representatives.

A precise specification and more detailed recommendations are left for further study after agreement on the basics has been reached.

5. Message Encapsulation

Basically, the aforementioned definition of a demultiplexing scheme for bundles based upon the (DTN) URI is sufficient for disambiguating bundles targeted at different applications. However, numerous (text-based) protocols in the Internet follow common conventions, such as using RFC2822-style headers and MIME encapsulation, which provide the benefit of code re-use across different applications. Therefore, it may be worthwhile discussing these conventions and defining a common encapsulation format for this class of applications. Currently, the set of text-based application protocols making use of similar message structures include (but are not limited to): electronic mail [3], HTTP [10], RTSP [9], and SIP [11]. Furthermore, numerous XML-based protocols (such as Jabber) could potentially benefit from a common encapsulation format.

The text-based protocols following the RFC2822 conventions usually consist of three parts: a start line, a sequence of message headers, and an optional message body. The message headers are subdivided into protocol header pertaining to the application protocol and entity headers describing the message body. The message body MAY be further encapsulated using the Multipurpose Internet Mail Extensions (MIME) [RFC2045, RFC2046] and MAY comprise multiple parts (using MIME multipart).

Other protocols are for further study.

5.1. Encapsulation Formats

The message body of a text-based message following one of the aforementioned protocols is usually identified by means of the Content-Type: entity header, but similar media type definitions also exist for the entire messages, e.g., message/rfc822, message/http, message/sip [http://www.iana.org/assignments/media-types/message/].

This leads to three alternatives for message encapsulation:

1. The application message is placed into the DTN bundle without further designation or encapsulation. In this case, the bundle content is implied from the EID and no further designation is provided.

Example:

```
[Primary bundle block]
[Bundle payload block
HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 13

Hello, world
]
```

2. The application message is encapsulated in a MIME envelope which includes an explicit Content-Type denoting the message contents as well as possibly other parameters (such as content transfer encoding). This supports including additional entity headers in the MIME envelope.

   Example:

```
[Primary bundle block]
[Bundle payload block
Content-Type: message/http
Content-Length: ...

HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 13

Hello, world
]
```

3. The message is encapsulated without the 'overhead' of a text-encoded MIME header but a separate header extension block is provided to include a content designation if desired. This field could simply indicate a MIME type (thus sharing the same registry), which may be preferred over setting up and maintaining yet another content registry. Optionally, additional MIME attributes might be included (is there value?). Protocols will be expected to specify default content types in which case this block would not need to be included, thus avoiding overhead for the default operation.

```
Example:

[Primary bundle block]
[Bundle payload identification block
 content-type: message/http]
[Bundle payload block
Content-Type: message/http
Content-Length: ...

HTTP/1.1 200 OK
Content-Type: text/plain
Content-Length: 13

Hello, world
]
```

Approach (1) is the simpler message encapsulation with less overhead and leads to slightly easier processing. However, it also precludes further operations such as encryption, authentication, or compression of the complete message. If such mechanisms were to be applied, the EIDs need to be used for demultiplexing (just as different port numbers are used for HTTP and HTTPS).

Approach (2) incurs more overhead (some 50 bytes minimum) but provides explicit content description and thus allows for demultiplexing different message formats.

Approach (3) incurs less overhead than (2) while maintaining the explicit content description and thus allows for demultiplexing different message formats.

Allowing multiple encapsulation formats, however, may increase the risk of non-interoperability (e.g., if a sender is not aware that the transfer encoding or compression used is not understood by the receiver). Such issues may be expensive or impossible to repair in DTN environments due to the potentially limited degree of interactivity. Application protocol encapsulation specifications would need to clearly address these issues and define mandatory to implement formats.

If the message content format shall be made explicit, as per (3), a dedicated extension block could also be used for this purpose (expanding further on the one proposed in section 6). The pros and cons of replicating such a description at the DTN layer will have to be investigated, depending on how many parameters will be needed to fully describe the bundle contents adequately.

In either case, allowing for an explicit identification of the

respective contents will enable distinguishing between a single contained resource and set of aggregated resources (using, e.g., multipart/mixed or multipart/related), e.g., multiple objects required to display a web page [7].

An open question is whether such a multiplexing of multiple closely related resources should happen within a single bundle payload, should occur by combining multiple payload blocks or bundles, or simply by using a batch of otherwise unrelated bundles. Or such a decision could be left entirely to the application-specific payload definition.

5.2. Security

Application protocols communicating via the bundle protocol may make use of the bundle security mechanisms (e.g., for end-to-end protection). In some cases, however, it may be desirable to maintain the application layer security properties (e.g., a signature on a piece of contents) beyond the bundle transmission and reception process. In such cases, security mechanisms should be applied above the bundle layer and attached to the payloads rather than to the encapsulating bundle protocols.

For the protocols described above, one viable option is using S/MIME [4] [5] as lasting encapsulation format. Depending on the intended semantics, either the entire application message or just the message body may be protected.

Note that for optionally protecting the entire application message, one of the enhanced encapsulation formats (2) or (3) from section 5.1 needs to be used.

6. Application Hints Extensions

As bundle agents store, possibly carry, and forward bundles from different applications, the nodes may carry these bundles for an extended period of time. In environments with frequent disconnections (and possibly short connectivity times), significant queues may build up in a bundle agent that may not be easily worked off during a single or a contact or contact time. This requires

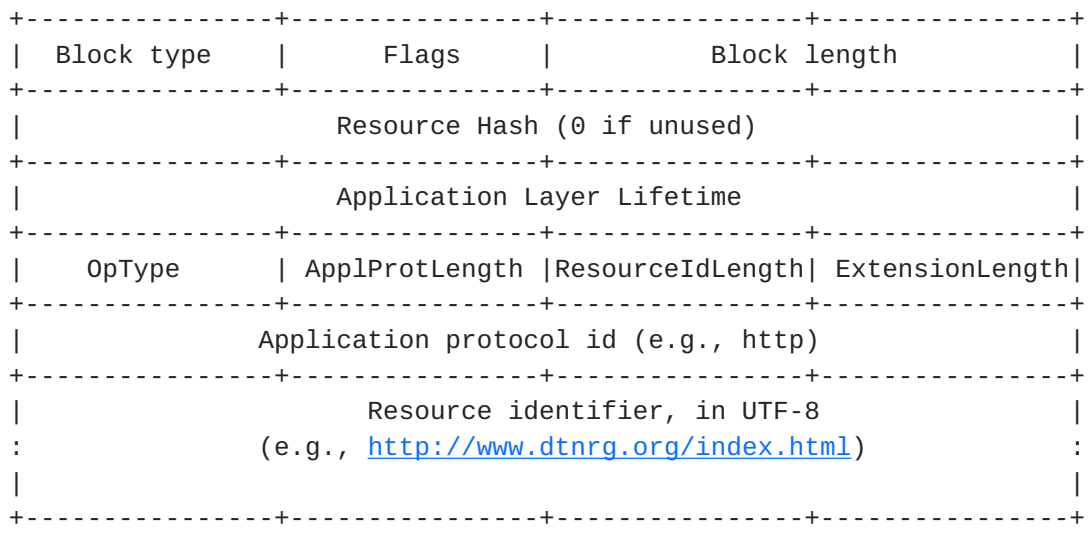
- a bundle agent to drop queued bundles if new ones come in,
- scheduling bundles for (re-)transmission to the next contact, and
- decide for which bundles to accept custody.

These decisions are taken in conjunction with the specific routing protocols and forwarding algorithms in use: these may demand replicating a bundle a certain number of times or transmitting it just once, determine to which contact(s) to forward a bundle, and define the order of deletion (e.g., earliest vs. latest expiry time, size-based). For infrastructure nodes dealing with random bundles or in closed environments with more or less 'equal' bundles, such mechanisms---as they should typically also be applied in IP networks---are just fine.

This neutral treatment can be questioned, however, when dealing with personal mobile devices running heterogenenous applications for two reasons:

- Mobile nodes may prefer to utilize local resources (power, CPU, memory) for bundles from those applications they are running by themselves. Similarly, requests and responses for application protocols may be treated differently. Both parameters could be made explicit in a block header.
- Beyond this, mobile devices may improve their service quality (e.g., response time, request completion probability) by employing cooperative caching mechanisms that leverage messages stored in bundle agents [OP2006]. For this purpose, a bundle agent would benefit from an identification of the resource contained in the bundle and its 'cache lifetime' (rather than its bundle lifetime).

To allow for such application-aware processing, the following extension block ("Application-Hints") is defined. (Note that draft-symington-dtnrg-bundle-metadata-block-00 defines a related metadata block, which is rather complementary as it describes a bundle while the Application-Hints defined below identifies a bundle and its properties.)



The Block type value for the Application Hints block is TBD.

The Flags fields specifies the Block processing fields are per section 4.3 of the bundle protocol specification [2]. The flags MUST be set as follows:

0 - TRUE. Block must be replicated in every fragment.
1 - FALSE. Transmit status report if block can't be processed.
2 - FALSE. Delete bundle if block can't be processed.
3 - As appropriate. Last block.
4 - FALSE. Discard block if it can't be processed.
5 - As appropriate. Block was forwarded without being processed. Should initially be set to FALSE.
6 - FALSE. Block contains an EID-reference field.

Block length: This field specifies the length of the block. This is an SDNV and is represented as a 16-bit value only for convenience of the presentation.

Resource hash: A SHA-1 hash of the resource contained (if any) excluding all headers to allow for efficient comparison.

The Application Layer Lifetime includes the validity of the resource (if any) using the same format as the bundle protocol. A value of zero specifies that the lifetime is unknown. This is an SDNV and is shown in the above figure as a 32-bit value only for convenience of the presentation.

The OpType is an 8-bit field indicating the application-specific operation type. The MSB indicates whether or not a resource is

included in the bundle. The following values are defined:

0x00 - Unknown / other / not specified, no resource included
0x80 - Unknown / other / not specified, resource included
0x01 - Request, no resource included
0x81 - Request, resource included
0x02 - Response, no rescource included
0x82 - Response, rescource included
0x03 - Unconfirmed event, no resource included
0x83 - Unconfirmed event, resource included

Further values are to be defined. The hint whether or not a resource is included shall give an indication to an intermediate bundle agent about the potential 'value' of this bundle.

The ApplProtLength indicates the length of the included application protocol identifier. A value of zero indicates that no protocol identifier is present. This is an SDNV and is represented as an 8-bit value only for convenience of the presentation.

The ResourceLength indicates the length of the included resource value (e.g., an application-layer message id or an HTTP URI). A value of zero indicates that no resource identifier is present. This is an SDNV and is represented as an 8-bit value only for convenience of the presentation.

The ExtensionLength indicates the length of an extension following the resource identifier. Details TBD. A value of zero indicates that no extension is present. This is an SDNV and is represented as an 8-bit value only for convenience of the presentation.

The application protocol id field is used to encode the protocol as a URI scheme. The resource should be a full URI or URN including the respective scheme, so that both the application protocol and the actual resource can be derived from this field.

The Resource value field is used to encode the resource contained in the bundle in UTF-8. The resource should be a full URI or URN including the respective scheme, so that both the application protocol and the actual resource can be derived from this field.

This document does not specify specific treatment of the Application-Hints block, but rather suggests providing such hints as a means for service differentiation in resource-constrained nodes. Drawing conclusions for local treatment is left to the implementation. (This entire section may ultimately be moved to a different document.)

7. Security Considerations

Any kind of application-specific tagging of contents (as, e.g., defined in section 6) is subject to forgery. Applications may attempt to guess for which hints they might obtain better service and fake the field to this end. As long as intermediate nodes exhibit heterogenous behavior (so that a malicious node cannot tell when it may receive better service), this issue is partly mitigated. Trustworthy identification of the sender will allow an intermediary to decide whether to take such application-hints into account or not.

Further issues are TBD.

8. References

8.1. Normative References

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

[2] Scott, K. and S. Burleigh, "Bundle Protocol Specification", draft-irtf-dtnrg-bundle-spec-10 (work in progress), July 2007.

[3] Resnick, P., "Internet Message Format", RFC 2822, April 2001.

[4] Ramsdell, B., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Certificate Handling", RFC 3850, July 2004.

[5] Ramsdell, B., "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.1 Message Specification", RFC 3851, July 2004.

[6] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, April 2007.

8.2. Informative References

[7] Palme, F., Hopmann, A., Shelness, N., and E. Stefferud, "MIME Encapsulation of Aggregate Documents, such as HTML (MHTML)", RFC 2557, March 1999.

[8] Handley, M. and C. Perkins, "Guidelines for Writers of RTP Payload Format Specifications", BCP 36, RFC 2736, December 1999.

[9] Schulzrinne, H., Rao, A., and R. Lanphier, "Real Time Streaming Protocol (RTSP)", RFC 2326, April 1998.

[10] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[11] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[12] Eddy, W., "Architectural Considerations for the use of Endpoint Identifiers in Delay Tolerant Networking", draft-eddy-dtnrg-eid-00 (work in progress), May 2006.

[13] Westerlund, M., "How to Write an RTP Payload Format", draft-ietf-avt-rtp-howto-02 (work in progress), July 2007.

Authors' Addresses

Joerg Ott
TKK Netlab
Otakaari 5A
Espoo 02150
Finland

Teemu Kaerkkaeinen
TKK Netlab
Otakaari 5A
Espoo 02150
Finland

Mikko Juhani Pitkaenen
Helsinki Institute of Physics (HIP) Technology Programme

Full Copyright Statement

Intellectual Property

Acknowledgment

Funding for the RFC Editor function is provided by the IETF Administrative Support Activity (IASA).