Moving data in DTNs with HTTP and MIME

Making use of HTTP for delay- and disruption-tolerant networks with convergence layers

Lloyd Wood, Peter Holliday and Daniel Floreani
London, England;
Brisbane and Adelaide, Australia
lloyd.wood@yahoo.co.uk,
pho@tpg.com.au, daniel.floreani@internode.on.net

Ioannis Psaras
Centre for Communication Systems Research
University of Surrey
Guildford, England
i.psaras@surrey.ac.uk

***Abstract*—Multipurpose Internet Mail Extensions (MIME) provide a simple way to describe the type of data sent and its use. Email and the web use MIME extensively to carry different files. The Hypertext Transfer protocol (HTTP) is universally recognized as a straightforward method to carry MIME objects as binary streams. Taking MIME and HTTP as a starting point, and adopting the well-understood need for different 'convergence layers' to carry HTTP in different challenged environments where TCP may not be suitable, this paper outlines work in progress to run HTTP over different transports, and how this can be used to create a simple, yet powerful, approach to relaying content in delay- and disruption-tolerant networks (DTNs).**

Keywords-HTTP, MIME, Delay-Tolerant Networking, DTN.

I. INTRODUCTION

The most popular uses of communication on the Internet – email, the web and the applications built using web technologies – owe their success to their ability to easily identify files and move copies around. This ability was given to email with MIME attachments [1], and the web and HTTP [2] reuse MIME with great success in parallel, with migration of MIME content between the two in the web so that *e.g.* web content can also be sent and handled by mail agents [Fig. 1].

As networking expands from fixed computers into the 'edge cases' of ad-hoc networks, or 'delay- and disruption-tolerant networks,' whose requirements for routing and transport are more demanding than those of the terrestrial Internet, the ability to access, move and identify files still matters; MIME can still fill a useful role in this domain, and identifying content for use with applications still matters.

In these challenged networks where permanent end-to-end connectivity is never guaranteed, different transport protocols, other than the Transmission Control Protocol (TCP) of the traditional Internet [3], are needed for local network conditions. When TCP fails due to long delays or disruptions in end-to-end connectivity, much of the traditional Internet infrastructure that relies on TCP will also fail to work.

However, HTTP is intended to be independent of TCP [2] and can still be used as a 'shim layer' to the local transport, providing a consistent interface for high-level applications to move and identify files with MIME via a hop-by-hop object exchange protocol.

9781-4244-3941-6/09/$25.00 ©2009 IEEE

We describe ongoing efforts to demonstrate the independence of HTTP from TCP by using HTTP over other transport protocols. These efforts move towards making HTTP a layer in its own right in the network stack. Using transport protocols suitable for DTN environments makes HTTP over those transports more useful for challenged networks. We leverage this by proposing simple additions to HTTP to create an 'HTTP-DTN' variant that can be used for delay- and disruption-tolerant networking.

II. DECOUPLING HTTP FROM TCP

HTTP is specified as a generic, stateless application level protocol independent of any transport protocol [2]. It can be thought of a session layer, running over a transport layer providing reliable delivery of the HTTP stream. Desire to separate HTTP from its traditional transport of TCP is increasing. For example, the Stream Control Transmission Protocol (SCTP) [4] offers a feature-filled superset of TCP's capabilities, and can carry HTTP [5]. There are many networking environments where TCP is not suitable, or is not implemented in small embedded computers, used for *e.g.* sensor networks, yet HTTP's simple text headers can still be used to identify, and to transfer, data [6]. For local network conditions where TCP or an equivalent such as SCTP is not suitable, an alternate transport layer with new behaviour, such as *Saratoga* [7], or even an HDLC bitstream, can replace TCP.

HTTP requires a minimum of reliable streaming that can be used to provide ordered delivery to the application; it is up to the local transport layer in the local subnetwork to provide or enhance that reliable streaming. For the examples given above, TCP or SCTP are used to carry HTTP over the congestion-sensitive public Internet, while *Saratoga* might be used across dedicated private links where congestion is not a concern.

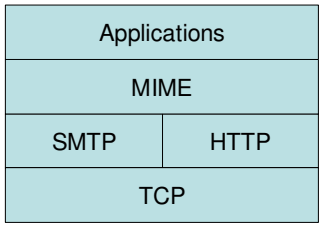

Figure 1. MIME use in the web and email

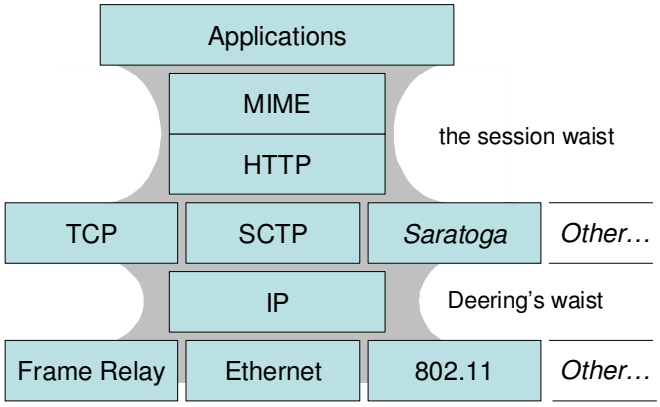

Figure 2. the protocol waists in the hourglass

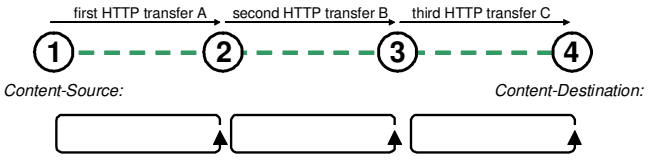

Figure 3. HTTP-DTN transfers end-to-end

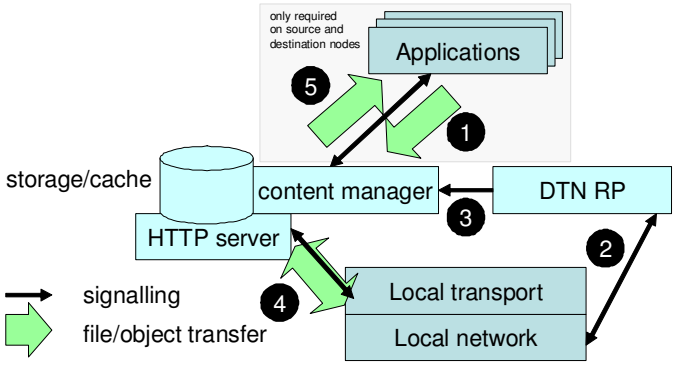

Figure 4. Internal operation of a HTTP-DTN node

At the packet level, IP, the Internet Protocol, has been described as 'the waist in the hourglass' [8] – what is above and interfaces with IP can be changed in a modular network stack, what is below and interfaces with IP can be changed, but provided the new elements continue to interface to and work with common paradigm of IP, the hourglass remains complete and the stack remains functional. Here, IP is a *network waist*.

At the file/object level, HTTP and MIME together form another *session* waist in the hourglass [Fig. 2]. HTTP is well-understood by applications, just as IP is well-understood by networks, encouraging its use. The applications can vary. The protocol providing the transport stream can be changed to suit local conditions, just as the link protocol is changed.

Being identified as a 'protocol waist' in this way indicates that HTTP can be regarded as a separate layer in the network stack in its own right. Given the prevalence of IP in many networks, it is likely that two popular waists (the popular layers that other layers interface to) exist: these are IP and HTTP. These can be used separately, but will most commonly be used together. The transport protocol and link protocol for the physical environment that IP and HTTP are used over vary more, as they depend on local conditions and requirements.

MIME and HTTP are grouped together here because MIME use for HTTP is described as part of the HTTP/1.1 specification [2], rather than being entirely consistent across both email, which MIME was first developed for, and the web.

III. ADAPTING HTTP FOR DTN

A simple approach leverages existing standards, using the HTTP as a transport-layer-independent 'session layer' between each two communicating delay-tolerant networking (DTN) nodes [9]. Rather than an end-to-end communication across a path as in the traditional Internet, this use of HTTP is hop-by-hop between communicating peers [Fig. 3].

HTTP runs over its 'convergence layer' – that is, a transport layer – suitable for local network conditions between each HTTP-capable node. New *Content-Source:* and *Content-Destination:* headers are added to HTTP. These provide late-binding endpoint identifiers, as the local Host: information is now only between peer HTTP nodes on the local subnet.

It is worth noting that *Content-*:* headers are treated specially by HTTP. HTTP servers must reject transfers with unknown Content- headers. Adding these two new headers effectively creates a separate DTN network that will not interact with or affect existing traditional web use of HTTP, as these 'HTTP-DTN' queries will be rejected by traditional HTTP servers unfamiliar with these headers.

Internal operation of the DTN node is shown [Fig 4]. We assume that an epidemic DTN routing protocol (RP) is used to manage connectivity [10]. Local applications interact with the HTTP server and its cache to store and retrieve MIME objects either by controlling simple HTTP PUT/GET or some other content management, *e.g.* WebDAV, SOAP, or AXL (1). Neighbour discovery and authentication tells the DTN RP of a new neighbour (2). The RP tells the content manager (3). An HTTP-DTN session with the neighbour exchanges data in caches. MIME objects can be pulled (GET or HEAD) or pushed (PUT or POST) from one node to another (4). Any received MIME objects with *Content-Destination:* matching the local node are handed to local applications, if any (5), while other objects are cached for the next hop. A series of separate HTTP transfers between peer nodes completes the end-to-end delivery of objects for distant nodes. The nodes act as caches of MIME content in transit between relaying HTTP transfers. This store-and-forward caching model differs from the intercept-driven proxy caching used in the World-Wide Web.

HTTP does not contain timers itself, so using HTTP with long-delay networks is possible. HTTP can work over extremely long distances provided that the transport protocol underneath it tolerates those distances; neither TCP nor SCTP are likely to be suitable. Supporting long distances requires features from HTTP/1.1: persistence to reuse open transport connections, pipelining to send multiple items without first receiving a response, and unidirectional PUTs to be able to send data blindly without acknowledgement if required.

IV. NAMING AND ADDRESSING

Delay and disruption-tolerant ad-hoc networks with mobility and without end-to-end connectivity pose challenges for routing, naming and addressing. Resolving named destinations can no longer be done once early with a lookup, as in the Domain Name System (DNS), but needs to be done by every caching node to determine where the destination might be and the current likeliest path towards it.

As a result, there is a need for applications to understand new addressing and namespace paradigms that can be used with HTTP. Fortunately, as HTTP's headers are in text, replacing textual IP or DNS addresses with other late-binding addresses, for other namespaces and routing, is straightforward.

Allowing an application to explicitly specify the desired transport and interface to use in the URI for a program to interpret when executing HTTP GETs or PUTs is also useful – particularly when a choice of mechanisms and interfaces are available and a selection must be made, and infrastructure such as DNS is not present so cannot be queried for advice [10].

V. A WORKED EXAMPLE OF HTTP-DTN USE

This summarises an HTTP-DTN file delivery, based on the ad-hoc temporary connections between nodes shown previously [Fig. 3]. This simplistic example is hypothetical, but is illustrates use of different technologies in a modular fashion.

Here there are three separate, disjoint HTTP/1.1 transfers, with their own reliable transport streams with control loops and acknowledgements, to describe in sequence. These examples are deliberately given illustrating the different ways that HTTP-DTN can transfer data.

An application at Node 1 places content into its local storage, for delivery to Node 4. Delivery is via Nodes 2 and 3, where the content is stored along with metadata on original source, final destination, timestamp, *etc*. before being relayed onwards.

The numbers given to the nodes are placeholders for locally-understood textual representations of addresses. Those addresses would be understood by the local DTN routing protocol running at each node, with separate exchanges of routing information and forwarding decisions that are not described here. These HTTP transfers are now described.

A. Transfer from Node 1 to Node 2

A wireless point-to-point link is established, using IPv4 addresses in the reserved 169.254.0.0/16 subnet [12]. Addresses are auto-allocated within that space [13]. Another address allocation method, *e.g.* static, or based on internal mechanisms, may be used. HTTP is transported over *Saratoga* streaming, after an exchange of *Saratoga* beacons advertising the nodes and Saratoga metadata setting up the streaming connection [7]. Node 1 sends at least these HTTP headers:

```
PUT content HTTP/1.1
Content-Source: <1>
Content-Destination: <4>
Host: <2>
Date: <original date of file>
Content-MD5: <digest>
Content-Length: <length>
Content-Type: <MIMEtype>

<content body>
```

Node 2 receives the content successfully and then responds:

```
HTTP/1.1 200 OK
Date: <current time>
```

B. Transfer from Node 2 to Node 3

Node 2 is dual-stack, while Node 3 is IPv6 only. A wireless link is established, using auto-allocation of IPv6 addresses in fe80::/10 [13]. The epidemic DTN protocol learns of the new neighbour and signals the cache manager to begin an HTTP-DTN session. The HTTP request is sent over SCTP [5], which Node 3 supports and tries first. Persistent connections, reusing the open transport stream for more transactions, are the default in HTTP/1.1. This allows onward forwarding of cached content from different sources in a single established transport stream.

Here, Node 3, the local destination, requests any content to transfer from Node 2, which is carrying our content from Node 1. Node 3 requests any content for forwarding from Node 2:

```
GET * HTTP/1.1
Host: <2>
Date: <current time>
```

Node 2 receives the request successfully and then responds:

```
HTTP/1.1 200 OK
Content-Source: <1>
Content-Destination: <4>
Date: <original date of file preserved>
Content-MD5: <digest preserved>
Content-Length: <length>
Content-Type: <MIMEtype>

<content body>

HTTP/1.1 200 OK
Date: <original date of second file>
Content-Length:
Content-Type:

<second content body destined through 3>
```

C. Transfer from Node 3 to Node 4

A shared wireless link is joined, using auto-allocation of IPv6 addresses in the fe80::/10 subnet [12]. After neighbour discovery, HTTP is sent over TCP [3]. Node 3 sends:

```
PUT content HTTP/1.1
Host: <4>
Content-Source: <1>
Content-Destination: <4>
Date: <date of file preserved>
Content-MD5: <digest preserved>
Content-Length: <length>
Content-Type: <MIMEtype>
```

```
<content body>
<any other headers and content to relay>
```

Node 4 receives the content successfully and then responds:

```
HTTP/1.1 200 OK
Date: <current time>
```

Node 4 keeps the original data with *Content-Destination: 4,* and hands it off to a local application selected using the MIMEtype in *Content-Type:*. Other data with different destinations is cached ready for further forwarding when routing suggests that a path to the destination is possible.

Although each transfer in this example relies on IP underneath the transport stream, the IP address only has local significance for the link as a well-understood protocol waist. IP routing is not required. If IP routing was used in a larger multiple-link subnet between two HTTP-DTN nodes, it would be unlikely to interact with routing used on other subnets, or with the different routing overlay interconnecting DTN nodes.

This example has copied a fixed file of known length. Using HTTP to transfer other resources of unknown length using the 'chunked' encoding is also possible. We are examining implementing the additional HTTP-DTN headers using the HTTP/1.1 functionality in Python.

VI. Comparing HTTP-DTN with the Bundle Protocol

HTTP provides the ability to easily transfer content identified by MIME. This provides useful content identification and signalling of which application to use that we have previously identified as missing from the Bundle Protocol. HTTP-DTN shares some of the same problems that we have recognised for the Bundle Protocol [14]:

- the assumption of synchronized timekeeping across all nodes that makes *Date:* timestamps and *Expires:* headers useful.
- the need to ensure end-to-end reliability of headers.
- Handling fragmentation imposed when a link drops, and delivery and reassembly of fragments taking separate paths in the network. Handling fragmentation is easiest when contact times and the amount of data that can be transferred to the next hop are known in advance (so-called *proactive fragmentation*). HTTP's *Content-MD5:* digest [15] is useful for checking successful reassembly of fragments.
- A way to indicate maximum accepted payload size that can be carried and stored is also needed.
- Security has been a focus of Bundle Protocol research. HTTP supports a number of security protocols that can be evaluated for suitability for reuse in unusual DTN conditions. However, as they are, these protocols would span separate links or subnets between HTTP-DTN peers, rather than going end-to-end across a disconnected path. DTN networks in general pose problems for key management.
- Routing, addressing and naming are not easy for DTNs.

However, by leveraging existing well-implemented and well-understood technologies, HTTP-DTN has benefits in being very easy to specify, understand, and implement.

Usefully, HTTP's headers are specified in text, and can be easily changed or added to with newly-defined headers. This is an advantage over any custom binary format, such as the Bundle Protocol, that is difficult to extend and modify in implementations. Human-readable communications tend to outlive even well-specified binary equivalents, *e.g.* the textual Rich Text Format and XML for word-processing documents.

The Bundle Protocol's textual Endpoint Identifiers (EIDs) could replace DNS names or IP addresses. This allows HTTP to leverage and interoperate with any addressing infrastructure built to support the Bundle Protocol, which can be replaced.

Conclusions

Work is in progress to separate HTTP from the underlying transport stream. This will make HTTP a 'waist in the hourglass' and layer in its own right. Applications should be able to work with HTTP regardless of the transport protocol underlying HTTP – and work to decouple HTTP from TCP and use HTTP with other transports is ongoing to enable this.

Using HTTP/1.1 to move content around delay- and disruption tolerant networks is feasible, requiring only that HTTP work over transports tolerant of DTN networks, with the addition of a few extra headers to support forwarding of MIME content to destinations across separate HTTP transactions. The resulting 'HTTP-DTN' warrants further study.

References

[1] N. Freed and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One," RFC 2045, November 1996.

[2] R. Fielding *et al*., "Hypertext Transfer Protocol -- HTTP/1.1," RFC 2616, June 1999.

[3] J. Postel (*ed.*), "Transmission Control Protocol," RFC 793, Sept. 1981.

[4] R. Stewart (*ed.*), "Stream Control Transmission Protocol," RFC 4960, September 2007.

[5] P. Natarajan, P. Amer, J. Leighton and F. Baker, "Using SCTP as a Transport Layer Protocol for HTTP," work in progress as an internet-draft, draft-natarajan-http-over-sctp, March 2009.

[6] E. F. Sadok and R. Liscano, "A Web Services Framework for 1451 Sensor Networks", IEEE Instrumentation and Measurement Technology Conference (IMTC) 2005, pp. 554–559, Ottawa, 16-19 May 2005.

[7] L. Wood *et al*., "Saratoga: A Scalable File Transfer Protocol," work in progress as an internet-draft, draft-wood-tsvwg-saratoga, May 2009.

[8] S. Deering, "Watching the Waist of the Protocol Hourglass," keynote, IEEE International Conference on Network Protocols (ICNP), Austin, Texas, October 1998.

[9] L. Wood and P. Holliday, "Using HTTP for delivery in Delay/ Disruption-Tolerant Networks," work in progress as an internet-draft, draft-wood-dtnrg-http-delivery, May 2009.

[10] Z. Zhang, "Routing in intermittently connected mobile ad hoc networks and delay tolerant networks: overview and challenges," IEEE Communications Surveys & Tutorials, Vol. 8, Issue 1, pp. 24-37, 2006.

[11] L. Wood, "Specifying transport mechanisms in Uniform Resource Identifiers," work in progress as an internet-draft, draft-wood-tae-specifying-uri-transports, May 2009.

[12] IANA, "Special-Use IPv4 Addresses," RFC3330, September 2002.

[13] S. Cheshire, B. Aboba and E. Guttman, "Dynamic Configuration of IPv4 Link-Local Addresses," RFC 3927, May 2005.

[14] L. Wood, W. M. Eddy and P. Holliday, "A Bundle of Problems," IEEE Aerospace conference, Montana, March 2009.

[15] J. Myers and M. Rose, "The Content-MD5 Header Field," RFC 1864, October 1995.