DataRouter: A Network-Layer Service for Application-Layer Forwarding

Joseph D. Touch, Venkata Pingali
{touch,pingali}@isi.edu
USC/Information Sciences Institute

May 9, 2003

Abstract – *DataRouter forwards packets at the network layer using application layer tags, without requiring per-hop termination of transport protocols and the consequent reimplementation of transport services in the application layer. The DataRouter provides network delivery based on pattern matching and string replacement. It combines a byte string as a loose source route IP option tag and regular expression routing entries to provide a new network service. DataRouter tags have a variety of forms, including fixed-length with exact matches for distributed hash tables and variable-length with regular expression matches for URL redirection. Conventional routing protocols configure application-specific routing with only minor extensions. Tagged IPv6 packets traverse non-DataRouter routers transparently. On a platform capable of IPv4 packet forwarding at 310K packets/sec., an preliminary (IPv4) unoptimized FreeBSD DataRouter implementation forwards hash-match packets at up to 270K packets/sec. (87% of max.) and pattern-match packets 155K packets/sec. (50% of max.). DataRouter thus provides a viable, higher-performance alternative to application-layer implementation of forwarding, in a generic service more interoperable with existing network and transport protocols.*

I. Introduction

DataRouter is an open, generic string match and rewriting facility for Internet packets. It augments the traditional, numeric address in an IPv4 or IPv6 header with an application-provided string used as a variant of loose source routing. The result provides an integrated facility for content delivery networks (CDNs), resource discovery, and advanced overlay network architectures.

The difference between a conventional IP packet and a DataRouter IP packet is shown in Figure 1 (only the relevant fields of the headers are shown). The DataRouter packet includes an option field, akin to the existing loose source route (LSR) option in IPv4 or the router header option in IPv6 [10][19]. The option contains a byte string (or multiple byte strings) with tag information. As with LSR, the packet is forwarded using existing IP routing tables towards the destination IP address; once there, the string is extracted, matched, indexed, and the packet header rewritten to indicate the IP address of the next hop router in the DataRouter topology.

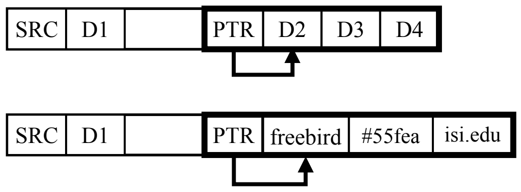

Figure 1 IPv4 Loose Source Route (top) vs. DataRouter (bottom) options

DataRouter provides an open platform for developing and deploying content delivery, peer-to-peer, and overlay networks that integrates with existing IP forwarding. This allows educational and research network architects to focus on the forwarding (matching and rewriting) and routing (table loading) protocols, and avoid the rediscovery and reimplementation of existing network-level capabilities or needing per-hop TCP connections which violate end-to-end semantics. Using sequences of DataRouter option strings, a single packet can traverse multiple, disparate CDNs, peer nets, and overlays without explicit application or inter-overlay gateways. The DataRouter capability can also be used to support both anycast and late-binding TCP.

DataRouter is a unified framework for integrating a variety of application-layer forwarding features implemented in a flexible network-layer mechanism. By using a single mechanism for a variety of services, DataRouter provides a core capability that can be tuned for network-layer performance. Finally, DataRouter packets are incrementally resolved into sequences of IP addresses, so they traverse conventional routers using existing IP forwarding between string matching routers, enabling partial, incremental deployment. A preliminary implementation indicates that DataRouting can be performed at rates competitive with that of conventional IP forwarding (50% of max.), four times faster than application layer forwarding mechanisms.

Background

The Internet forwards packets based on fixed endpoint identifiers, i.e., IP addresses. Routing uses these addresses to direct packets toward their destination, using longest-prefix match in forwarding tables, on tables that have been loaded either manually or by a routing protocol. The Internet currently supports a single, global address space, and a single, global set of forwarding tables.

Existing techniques to support additional matching schemes require separate distributed systems. Conventional Internet resource discovery uses an external table, specifically the DNS, to resolve names to IP addresses. As another example, Google is a central database that resolves text phrases to URLs, which include DNS names or IP addresses directly. More recent peer-to-peer architectures forward requests over application-layer tunnels (e.g., TCP connections) and use a distributed application to direct queries to a table [18].

Such services enable interesting and useful content-directed forwarding at the expense of violating the "end-to-end principle" [22]. In the Internet architecture, the network layer forwards packets, the transport layer maintains ordering and reliability (if desired, as well as congestion control), and only the application deals directly with the payload data. In a CDN, data-layer information is used for forwarding, e.g., peeking at the URL inside an HTTP request to route HTML requests over a slow pipe and JPG (image) requests over a fast pipe. CDNs can direct requests for bandwidth, cache aggregation, or policy-based routing.

However, in all cases the TCP connection must be terminated per-hop, in order to reassemble the packets sufficiently to recover the data stream; this necessitates application-layer mechanisms to ensure end-to-end reliability and resequencing. An alternative is to peek into packets and try to recover the data without terminating the connection, which can be challenging when packets take diverse paths or when the data is encrypted. Either case violates (or at least badly strains) the "end-to-end" principle.

DataRouter replaces CDN's external, application-layer mechanisms with an open, network-layer matching and rewriting service. End host applications can add a byte string to the network (IP) header as a new type of IP option, and that header is looked up and/or rewritten at intermediate hops, using a separate set of loadable tables. The result achieves integrated routing based on application data utilizing a unified network layer service, without requiring per-hop TCP termination or peeking at packet data.

Current Internet routing supports "loose source routing" in IPv4 and IPv6, in which packets are forwarded to a chain of explicitly-selected routers using an IP header option [10][19]. The packet contains a conventional source IP address, the IP address of a destination where LSR is performed, and the header option with a list of subsequent destination IP addresses. At each intermediate destination, the header IP destination field is exchanged with the address at the pointer in the option, and the pointer incremented (Figure 2). The process stops when the pointer exceeds the option length.

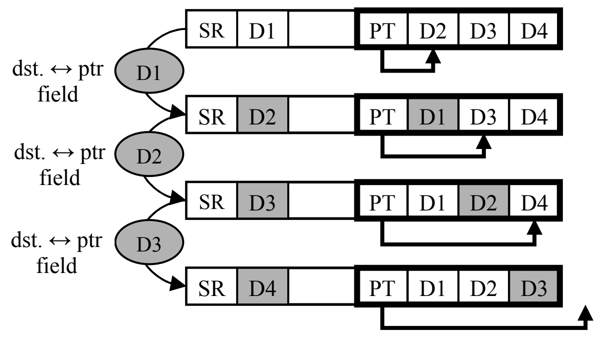

Figure 2 IPv4 Loose Source Routing – step by step

DataRouter extends LSR so that the chain can contain arbitrary application-configured strings such as DNS names, URLs, etc.. DataRouters lookup these strings to entries in a table that indicates the IP address of the next rewriting-router and rules for rewriting the string (if desired) (Figure 3). DataRouter thus augments conventional IP routing with a generic string substitution and match capability.

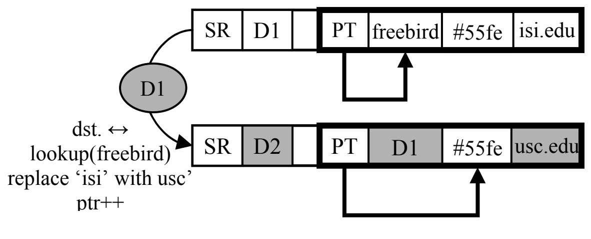

Figure 3 Single step of DataRouter option processing

DataRouter's LSR-like forwarding allows overlay networks to be incorporated in the base Internet architecture. This facilitates multi-overlay paths without the need for inter-overlay gateways. The string labels allow application-layer content-based routing without requiring connection termination at each hop, avoiding complications with end-to-end reliability and further facilitating the integrated use of various content delivery (a.k.a. distribution) systems (CDNs).

Current peer-to-peer and CDNs use application-layer tunnels between forwarding components, to allow the forwarder to access packet data [18]. For example, a URL CDN system requires a separate TCP connection per hop, so that the URL in the data stream can be viewed by the forwarder. DataRouter places that information in the IP packet header, making it accessible to the forwarder without requiring separate, hop-by-hop connections.

The result is more consistent with conventional network architectures, where forwarding uses packet header as context, and transport (TCP) connections provide end-to-end reliability. DataRouter thus provides content-based routing without violating the "end-to-end argument," or requiring separate, application-layer reliability mechanisms [22]. It can also be used to integrate DNS resolution and TCP connection establishment (SYN) phases, reducing connection latency and improving performance for short connections or anycast services [13][16].

Although there have been a number of new recent network architectures, both at the peer-to-peer and network overlay levels, incremental deployment and management of these systems has been examined in a limited way. Most systems assume that new capabilities are deployed at specific routers connected by tunnels at the application or network layer. DataRouter provides an alternative deployment environment in which new routing tables are loaded, but no new tunnels need to be created. This provides new opportunity, but also represents a paradigm shift for application-layer network architects; they focus on being routing protocol designers more than tunnel engineers.

Current inter-peer and inter-overlay communication requires gateways, explicitly deployed at key points in the architecture. DataRouter allows composition of inter-overlay paths by concatenating data tags in the IP option, providing new opportunity for more pervasive, flexible, and dynamic creation of heterogeneous routing paths.

Finally, DataRouter also supports both anycast and late-binding TCP [13][16]. Both capabilities merge address lookup with packet delivery. For anycast, the service (e.g., "printer") is the string in the anycast IP packet, and the initial destination is the first lookup node of an anycast database. A type of late-binding TCP can place the DNS name is the string in the SYN packet, and set the base IP destination address to the DNS server. This variant of TCP is related to dispatching HTTP requests within web server farms, as well as to reduce TCP connection establishment over fast links. Other support is required, e.g., to allow late resolution of port numbers (as discussed in Section V), but the DataRouter provides a key component of a solution.

II. DATAROUTER

DataRouter is a generic string match and rewriting capability at selected routers. It consists of the following components:

1. IP option structure
2. forwarding and rewriting tables at selected routers
3. a fixed set of matching algorithms
4. API for applications to write/read the IP option
5. API for routing algorithms to load the tables

The combination of these components provides a string match and string replacement corollary to the current IP forwarding and routing mechanism.

IP option structure

The current IPv4 Loose Source Routing (LSR) option (also called *Loose Source Route and Record*) consists of a tagged option entry with a pointer and length fields, followed by a sequence of IPv4 addresses; a similar option called a *routing header* (RH) exists for IPv6 [10][19]. In IPv4, the LSR option uses 3 octets for the LSR tag (0x83), the option length, and an octet pointer used to step through the addresses (Figure 4).

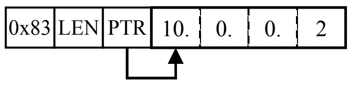

Figure 4 IPv4 LSR Option

In IPv6 (Figure 5), the RH option is indicated by a field in the previous option or base IP header, and the option consists of 4 octets of control information – the type of the next header (NH), the length of the option in 8-octet units (excluding the first 8 octets), the type of routing (e.g., 0 indicates LSR), and a counter indicating the number of unprocessed segments left (effectively a pointer), followed by the address list. Overall, the two options are essentially the same, except that in IPv4 the space for all options is limited to 40 octets, whereas in IPv6 there is no limit per se to option space.

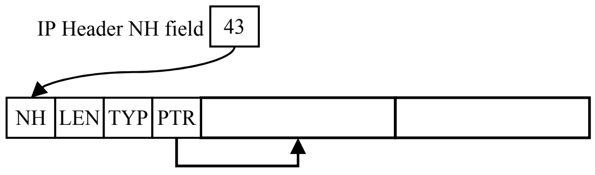

Figure 5 IPv6 Routing Header Option

In either IPv4 or IPv6 the packet is delivered to the destination address in the base IP header; upon arrival, the router writes address at the pointer in the header destination field, and the router's canonical address at the pointer in the option, and then increments that pointer. This serves the dual purpose of chaining delivery through a fixed sequence of routers, as well as recording the canonical address of the routers thus visited. Note that these routers need not be adjacent; they represent only specified addresses that must be visited, but there may be routers between which forward the packet via conventional processing.

DataRouter uses a similar structure, using a chain of labeled strings rather than numeric addresses. As with LSR, the DataRouter option contains a length and a pointer indicating the string to be manipulated at the next destination hop, in addition to the option tag itself (Figure 6, left). A list of string structures follow, where each string is tagged with a routing class, a matching and lookup algorithm, and the string's length (Figure 6, bold; string shown in shaded area). The routing class indicates which tables are used for matching and translation, enabling concurrent use by multiple routing systems.

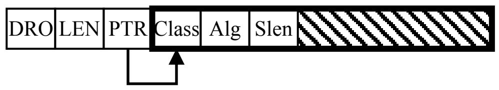

Figure 6 DataRouter Option

The structure shown in Figure 6 is for an IPv4 DataRouter option. In IPv6, the option is a variant of the existing routing header option (Figure 5), where the type (TYP) is DRO (DataRouter option). IP addresses resolved by patterns match the IP version of the base header, a requirement enforced by the routing protocol.

The fixed set of lookup algorithms is:

1. longest pattern match
2. exact match
3. longest prefix/suffix
4. "closest" match (fuzzy match)

The first two have been implemented and tested, as will be discussed in Section III. The addition of a cost function (for (4)) both increases the complexity of the implementation (and thus lowers its performance) and increases the potential for ambiguity. Note that (2) and (3) are special cases of (1), e.g., where all patterns are exact matches, and would be provided as specially-tuned alternatives.

The use of a set of predefined classes enables concise descriptions of various anticipated configurations. For example, the following examples define current forwarding methods, including a sample tag string, and a string-based routing class for each method:

a) DNS lookup

   lookup=#longest_suffix
   class=#DNS
   string=*joe.com*

b) URL redirection

   lookup=#exact
   class=#URL
   tag=*joe.com/apple*

c) Napster/Chord/CAN content-directed lookup

   lookup=#exact
   class=#MP3
   tag=*hash(song name)*

d) Google content-directed lookup

   lookup=#closest
   class=#WEBDBASE
   string="*Harry Potter movie*"

The use of different routing classes allows two schemes with the same lookup algorithm to utilize separate tables, even at the same router, e.g., b) and c) above. Lookup algorithms and classes are shown as constants (#), but are represented by numeric indices. Current IPv4 LSR capabilities can also be shown in this generic scheme:

e) IPv4 LSR

   lookup=#longest_prefix,
   class=#IPv4
   string=*10.0.0.2*

IPv4 has limited capability for such options, with only 37 octets of total DataRouter option payload possible (40 max., less 3 for the option tag), where each string requires an additional 3 octets of overhead. Optimizations may be possible, e.g., merging the string Class and Algorithm fields, to reduce this overhead, but it is clear that IPv4 DataRouting is constrained.

An IPv6 DataRouter option can be much larger. The existing RH option, which can be used for DataRouter (e.g., using Type=1), can be up to 2K octets per instance, and appears to be no limit to the number of RH options in a single packet. The total IPv6 option space can consume as much of the overall packet size as desired (64K conventionally, or 4G using jumbograms) [4].

Forwarding and rewriting tables

All DataRouter-capable routers include a separate set of forwarding/rewriting tables, to be matched by the strings in the DataRouter option. The set of tables for each class used by a particular option is indexed by the class identifier. Each class table entry consists of:

- **match field:** the field against which the string is matched using the lookup algorithm.
- **rewriting rules field:** a set of rules for rewriting this, or perhaps subsequent (but not antecedent) DataRouter option strings. In most of the examples shown above, the rule is "replace with the current router's IP address".
- **IP address:** the address of the next DataRouter in the path for this entry.

This set of rules provides a generic capability; the rewriting rules in particular augment the indexing capability to allow on-the-fly revision of subsequent DataRouter options.

Lookup algorithms

DataRouter includes a small set of fixed lookup algorithms. The objective is to provide a flexible and generic capability, not a complete programming environment. Perl-like patterns represent some of the more powerful descriptions, because they can find repeated strings, or context-based matches. More common usage will be dominated by the string structure:

- exact match for hashes
- longest prefix for IP addresses
- longest suffix for DNS names

URLs represent a special case, one where the rewriting rules may be especially useful. Consider http://www.isi.edu/touch/index.html, which benefits from successive DataRouter resolution:

1. longest suffix anchored at the first single "/" – once there, remove http://www.isi.edu/
2. longest prefix thereafter

In this case, DataRouter would rewrite the current option or insert a copy before the next string to be processed. Alternately, the component operations could be decomposed by the application.

IP option API

Applications need a mechanism by which to set the IP DataRouter option, and a way to read the option contents upon delivery. Unix *sockopts* provide this capability. For DataRouter, the options can be set per-packet, or per socket (per-association for UDP or per-connection for TCP). There may be further implications on the requirements for Internet hosts, as well as for the routing table values [5].

The DataRouter requires additional transport layer support for late binding [13]. Incrementally resolving strings into IP addresses is consistent with existing IP, but UDP and TCP include the final endpoint address in a loose source route list in the transport protocol processing. For TCP and optionally UDP, this affects the calculation of the transport checksum. For TCP, it also affects connection processing, because TCP expects to match returning SYN/ACKs with the emitted SYNs, based on addresses and ports. Existing solutions that support host mobility can be applied, notably the Host Identity Payload (HIP) protocol, which uses an intermediate header between the IP and transport protocol, to provides exactly the decoupling required [17].

Table loader API

The tables of a DataRouter-capable router need to be loaded by a routing protocol. This proposal does not address the routing protocol, as there are many to choose from, and it focuses instead on enabling the development of these protocols.

The current API for loading forwarding tables is a variant of the Unix *route* command called *droute*:

droute

class *class_id*

add *pattern dest*

[alg (longest|exact…)]

The default algorithm is "longest". "Dest" indicates the IP address of the next hop to use when this pattern matches. The current implementation supports *regexp* patterns [21]. Further details of the pattern are under development, including how best to indicate the following:

- match only & delete current string
- match & substitute on current string
- if matching current string, then substitute on all subsequent strings

III. PRELIMINARY RESULTS

A preliminary implementation of the DataRouter has already been completed in FreeBSD 5.0, using a new IPv4 option and UDP data packets [14]. It includes a preliminary API for inserting and reading options and configuring tables. It supports exact and longest suffix match, and was tested for the classes of

MP3 hashes and DNS names. This implementation consists of ~700 lines of kernel code and ~1,000 lines of application code for testing. This version consists of longest-pattern match lookup only (current string deleted), to indicate the upper-bound performance of an unoptimized system.

The results of preliminary tests indicate the utility of this mechanism. Both conventional DNS and peer-to-peer style MP3 hash lookups are supported using a single interface. The system avoids per-hop transport-layer tunnels and works as an intermediate step in a global Internet path. The data are summarized in Figure 7.

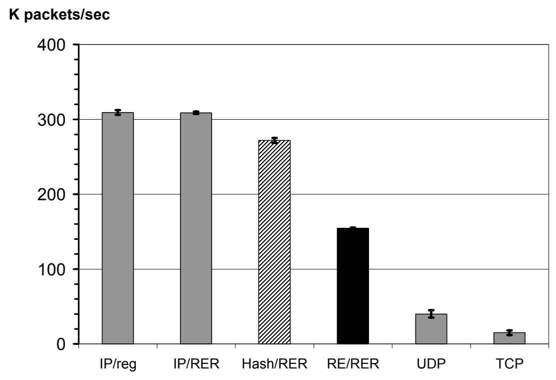

Figure 7 Comparative IPv4 forwarding performance (in K packets/sec)

On a dual-processor 2.4 GHz Xeon PC running the existing FreeBSD 5.0, IPv4 packets are forwarded around 310K packets/sec, indicated as "IP/reg" in Figure 7 (leftmost bar). Forwarding the same packets on a kernel with DataRouter extension support[1] (IP/RER) does not affect performance measurably. DataRouted packets forwarded based on an exact match of a 32-bit hash decreases performance to around 270K packets/sec (Hash/RER, striped), and forwarding based on a regular expression (in this case, "*.(isi|usc).edu") results in 155K packets/sec. (dark bar). These are simple baseline experiments, in which all packets for a test have the same header, and the forwarding tables have only one entry of each type (regular longest-prefix, hash, and regular expression). The graph shows averages of 10 1-minute runs over 64-bit/66 MHz PCI gigabit Ethernet connecting three machines (source, router, sink), with error bars indicating +/- 1 standard deviation.

Compare these results to application layer forwarding, also shown in Figure 7 (right two bars). Trivial application-layer UDP forwarding, using a single, default output route, runs at 40K packets/sec. on the same PCs. TCP forwarding is limited to the number of new connections per second, 15K connections/sec. when TCP TIME_WAIT states are discarded on close. Moving forwarding into the kernel avoids data copying across kernel-user boundaries, as well as reducing interrupt processing overheads. Although these rates could be increased with tuning, application forwarding still complicates end-to-end semantics for TCP connections.

These tests measured an IPv4 option; the ultimate goal is an IPv6 implementation. IPv6 provides additional option space and provides a safer environment in which to experiment with new options. IPv4 options should be ignored at intermediate hops, notably at routers forwarding DataRouter packets toward their next hop.

IPv4 packets with new, unrecognized options should be ignored (i.e., forward normally) at intermediate routers on paths between DataRouters [3]. Past experience in the Internet community deploying new options suggests caution, however, notably because options not already supported often divert packets from hardware 'fast-path' processing to outboard 'slow-path' software. Section V suggests a technique to overcome this potential pitfall.

The code can be transitioned to the core of the Internet if desired, using either PCs as buddy-routers or by integration into native routers. Because DataRouter provides CDN-like redirection, implementation in the Internet core is not necessary, and it may be more convenient to rely on edge-based DataRouter services for directing initial requests, where subsequent data connections can utilize conventional IP packets.

[1] i.e., capability present, but unused.

IV. Related Work

DataRouter extends the concepts of a number of peer, overlay, and alternative network architectures, unifying the generic capability believed to support many of these systems. It is a very specific capability, and though it could be deployed using programmable (Active) networks, it is more consistent with a static capability with dynamic configuration than a truly programmable system [25]. Further, it is distinct from most of these related architectures in being integrated (and relying upon) the underlying IP forwarding infrastructure. DataRouter augments IP routing with data routing, but does not replace it.

Peer networks

DataRouter is inspired by the use of application data for content-directed routing in peer networks [17]. Whether accessing URLs in an HTTP connection in a TCP stream, or hashes as used in CAN or Chord, these systems forward using packet data, rather than packet headers [20][24]. In some cases the data is extracted en-route, in other cases the hash is performed a-priori to provide a header destination address. The use of data for forwarding distinguishes them from VPN or overlay networks, which rely on conventional endpoint addresses.

The use of data for routing provides new capabilities, notably content-directed networks. These replace centralized databases for resource discovery, providing a distributed, scalable, self-organizing database. The cost is large, however – either an entire, separate topology must be deployed (CAN/Chord) or each hop must terminate the data (TCP) connection (to access the packet data properly). The former is cumbersome and prohibitive, and the latter violates the end-to-end argument, requiring separate application-layer reliability mechanisms on top of conventional transport protocols [22].

There are more recent systems which focus on the naming structure of CDNs (e.g., INS) or use CDNs for rendezvous-based communication (e.g., III, or *i3*) [1][23]. In both cases, as well as with other CDN systems (hash or string-based), DataRouter can provide a platform in which INS, *i3,* or other architectures can load the rewriting tables, allowing network-layer processing based on application-layer data, and avoiding the need for each of these (and other emerging) architectures to reimplement a network layer processing capability.

Overlay networks

Overlay networks are deployments of virtual infrastructure, using separate endpoint addresses, tunnels, and routes. They too are cumbersome to deploy, and require separate name-to-address mapping mechanisms to be useful. DataRouter provides overlay-like capabilities using the core Internet, replacing tunnels with data-directed loose source routing. LSR was abandoned as a mechanism to deploy new protocols in the early days of the M-Bone, because LSR-tagged IPv4 packets are processed inefficiently at every router hop [11]. IPv6 removes this impediment, such that LSR-tagged packets are handled differently (from non-tagged packets) only at hops where the LSR header is manipulated [10].

DataRouter builds on virtual networking systems such as the X-Bone, and the ways it is being applied to other domains, notably geographic overlays (GONet) [12][26]. It provides an integrated system for supporting multiple overlay systems, and internetworking among them. It also integrates with core Internet routing more directly than tunnel-based systems such as RONs [2]

Alternate network architectures

DataRouter allows the deployment of alternate network architectures, notably those that benefit from an index-based forwarding. It thus enables tests and incremental deployment of IPNL, TRIAD, Heaps, and Network Pointer architectures. It is fundamentally based on the Linda system's tuple-style message delivery, integrated with existing IP forwarding.

IPNL is a multi-level routing hierarchy, utilizing different forwarding tags at various routing levels [13]. DataRouter can be used as a platform for developing IPNL concepts, using sequences of DataRouter strings for the various IPNL forwarding tags. TRIAD similarly uses an alternate tag architecture, preferring DNS names to IP destination addresses; here again DataRouter option strings can be used to provide TRAID-like service [9]. Similar multilayer forwarding in Network Pointers, and Catanet can be supported [8][24][27].

Catanet first described the use of source routes and addresses having additional structure, albeit using different classes of more conventional IP addresses [8]. This concept is augmented to use pointers (Network Pointers) or heaps in newer proposals [6][27]. DataRouter is a more general variant of the use of multiple addresses, although currently assuming a linear structure. Instead of focusing on the semantics of the addresses (pointers are a form thereof), it focuses on generalizing the indexing and rewriting capability present in various forms in all these earlier or alternate proposals, specifically allowing the indexing to occur in the network on the path.

The use of a indexing for communication in DataRouter is inspired by the Linda distributed system, which used tuples for message passing [7]. DataRouter extends this concept with rewriting, and uses it for message delivery across the Internet rather than within a single system.

V. RELATED ISSUES

Preliminary implementation of the DataRouter option in IPv4 suggests that string matching can be done at reasonable rates. Ultimately, its use by application and protocol designers will determine its impact. There are a number of open issues in the current DataRouter research which are largely a matter of development. IPv4 transparency can be supported by a modified option format. Although the current version operates at reasonable rates, optimization may increase throughput substantially. Integration of the DataRouter option with existing routing protocols requires similar option extensions to those protocols. Support for late-binding requires endpoint identification similar to emerging standards, with interesting additional requirements. Even given those potential issues, the benefit to application protocol designers, notably avoiding the need to reimplement transport layer services in the application layer, is substantial.

IPv4 Transparency

New IPv4 options often present a challenge to the existing Internet infrastructure. As noted earlier, IPv4 routers are required to ignore options they do not recognize, so the presence of the new DataRouter option should not impact the forwarding of packets at conventional routers between DataRouters.

Unfortunately, experience shows that new options can affect packet processing. At best, non-compliant routers may process such packets on the 'slow-path' in software at substantially-reduced throughput. At worst, routers may drop packets with unrecognized options. This is more of an issue with the legacy IPv4 installed base; compliance with IPv6 router option requirements has been more closely scrutinized.

A viable alternative to the new option structure shown in Figure 6 is to masquerade as a conventional loose source route option (Figure 4), as shown in Figure 8.

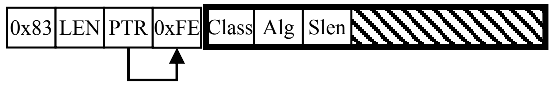

Figure 8 DR masquerading as a LSR

When the first byte of the destination address is 0xFE, the address is in the end of the reserved range, currently unspecified [15]. As such, it should currently never occur as a destination IP address, so its use as the first octet of a conventional LSR address field can be used to indicate that the following octets are a DataRouter field, shown outlined in bold.

A masquerading DataRouter option should be correctly ignored by legacy IPv4 routers when they are not the destination IP address in the base IP header. This constraint is satisfied by the DataRouter routing protocol. Because pattern routes cannot be installed on such a router, they should never be the destination of a DataRouter forwarding entry, and so should never end up being looked-up and inserted in the base IP destination field.

Additionally, some legacy routers may perform 'sanity' checks on LSR values, e.g., checking that the LEN is a multiple of 4 octets, or that the PTR is aligned on a word boundary. These can be accomplished with appropriate padding of the masquerading fields.

The overhead of the additional octet of overhead in a masquerading option can be reduced by the compression of other fields, notably the algorithm and string length. The string can be limited to 31 octets, so that the string length can be represented in 5 bits and 8 algorithms indicated in a compressed Alg/Slen field. This is reduces the per-string overhead to 2 bytes, but without major impact, because there remains only 36 octets for a string anyway.

Optimization

The current implementation uses the complete *regexp* library in an unoptimized kernel [21]. As noted earlier, there are some algorithms which to not benefit from the features in this complete library, e.g., exact match. Some algorithms may benefit from a subset of features, but do not need the full flexibility *regexp* provides.

An open area of research is to determine the smallest useful subset of *regexp*. Although this library is highly optimized, its flexibility renders it large and complex. More limited subsets may suffice for popular classes of DataRouter applications. A smaller set may provide further opportunities for optimization, as well as shrinking the working set of packet processing code, e.g., to reduce instruction cache thrashing.

Routing Protocol Integration

The current implementation uses a user command (*droute*) to set pattern-based routing table entries. Support for dynamic routing protocols would enable a distributed set of DataRouters to operate as an overlay network while avoiding tunneling complexities [26].

Current dynamic routing protocols exchange IP addresses, routing metrics, Autonomous System numbers (AS's), etc. Augmenting these protocols to exchange DataRoutes is primarily a matter of specification and implementation. Similar support needs to be added to routing analysis tools, the SNMP routing MIBs, etc.

Late-binding Support

Late-binding enables applications to use the DataRouter option to integrate string-based forwarding with existing transport protocols, such as TCP and UDP. These protocols often use the destination IP address in the transport protocol, e.g., for data checksums or state management, even for connections not yet established. When the last element in a DataRouter option is not an IP address (the typical case), the packet lacks sufficient information to compute the checksum or configure local protocol state.

Consider a conventional loose source route IPv4 packet, as shown in Figure 2. The packet is emitted with an initial destination (D1), and a list of LSR hops (D2, D3, D4). When the packet is first emitted, its transport layer checksum is computed using the final LSR hop address (D4), so that when the packet reaches its destination (at D4), the checksum computed by the receiver matches that in the transport header.

The challenge for DataRouter is similar to that for anycast or mobile IP networks [16]. In anycast, the destination IP address is not known when the packet is emitted, as for DataRouter. For mobile IP, the endpoint may be determined but its IP address may change during a connection. In the latter case, a shim-layer protocol, between IP and the transport protocol, can embed unique addresses that substitute for IP addresses in transport protocol processing, e.g., the Host Identity Payload protocol (HIP) [17].

In DataRouter, as in anycast, the destination is not known, so a nonce cannot be exchanged out-of-band or retrieved from a cache as with HIP. A potential solution may be to allow the source to specify the destination nonce HIP value.

For TCP, the destination address (and other info.) is also used to match incoming packets to connection state. In a similar fashion, HIP tags can be used as a replacement for the destination address. This allows TCP SYN packets to merge DNS resolution with connection establishment, as in IPNL [13]. The packet is emitted with a DataRouter string of the DNS name, towards the DNS server. The DNS resolves the IP address, and DataRouter rewriting redirects the packet towards the destination. Because DNS servers are often on the same path, if not ultimately very close to the resolved connection endpoint, this merging of DNS and TCP can remove a round trip time from exchanges with new sites.

Impact on End-to-End issues

As noted earlier, application layer forwarding has a negative effect on end-to-end protocol properties [22]. Connectionless (e.g., UDP) fragmented packets must be reassembled at the forwarding routers, and connections (e.g., TCP) must either be terminated at each hop or snooped and spoofed to reconstitute their internal data.

Encrypted data connections prohibit application layer forwarding, unless keys are distributed to all intermediate routers, destroying end-to-end security. When connections are terminated at intermediate hops, error detection and correction must be reimplemented at the application layer, to ensure end-to-end reliability.

DataRouter avoids this complexity, allowing the application to place forwarding data directly in the network-layer header. That data is then accessible by DataRouters, copied into fragmented packets, and not subject to data encryption.

VI. Conclusions

DataRouter provides a generic string matching and rewriting capability, enabling application-directed forwarding with network layer efficiency, yet without requiring extraordinary measures at the application layer. A preliminary IPv4 implementation demonstrates that DataRouting can operate at 50% of IP forwarding rates, 4 times faster than is possible at the application layer.

The DataRouter enables deployment of new forwarding services incrementally, forwarded like loose source routed packets over existing legacy Internet infrastructure. These new capabilities require modest extensions to existing transport protocol processing, akin to those already required for IP mobility and anycast. By providing an integrated, network-layer capability, DataRouter enables application and protocol designers to focus on the specific features of the system, rather than the details of the mechanism that provides it.

VII. References

[1] Adjie-Winoto, W., Schwartz, E., Balakrishnan, H., Lilley, J., “The Design and Implementation of an Intentional Naming System,” Proc. ACM SOSP (OS Review, V34 N5), Dec. 1999, pp. 186-201.

[2] Andersen, D., Balakrishnan, H., Kaashoek, M., Morris, R., “Resilient Overlay Networks,” Proc. 18th ACM Symp. on Operating Systems Principles (SOSP), Oct. 2001, pg. 131-145.

[3] Baker, F., “Requirements for IP Version 4 Routers,” RFC1812, June 1995.

[4] Borman, D., Hinden, R., Deering, S., “IPv6 Jumbograms,” RFC 2675, Aug. 1999.

[5] Braden, R., ed. “Requirements for Internet Hosts -- Application and Support,” RFC 1123, Oct. 1989.

[6] Braden, R., Faber, T., Handley, M., “From Protocol Stack to Protocol Heap – Role-Based Architecture,” Proc. HotNets-I, Oct. 2002, in ACM CCR, Jan. 2003, pp. 17-22.

[7] Carriero, N., Gelernter, D., “The S/Net's Linda Kernel,” ACM Transactions on Computer Systems (TOCS), V4 N2, Nov. 1986, pp. 110-129.

[8] Cerf, V., “The Catanet Model for Internetworking,” IEN 48, July 1978.

[9] Cheriton, D., Gritter, M., “TRIAD: A Scalable Deployable NAT-based Internet Architecture", Stanford Computer Science Technical Report, Jan. 2000.

[10] Deering, S., Hinden, R, “Internet Protocol, Version 6 (IPv6),” RFC 2460, Dec. 1998.

[11] Eriksson, H., “MBone: The Multicast Backbone,” Communications of the ACM, Vol.37, Aug. 1994, pp.54-60.

[12] Finn, G., Touch, J., “Network Construction and Routing in Geographic Overlays,” ISI Technical Report ISI-TR-2002-564, July 2002.

[13] Francis, P., Gummadi, R., “IPNL: A NAT-Extended Internet Architecture,” Proc. Sigcomm 2001, Aug. 2001, pp. 69-80.

[14] FreeBSD man pages, e.g., http://www.freebsd.org/

[15] IANA, “Special-Use IPv4 Addresses,” RFC3330, Sept. 2002.

[16] Johnson, D., Deering, S., “Reserved IPv6 Subnet Anycast Addresses,” RFC 2526, March 1999.

[17] Moskowitz, R., “Host Identity Payload Architecture,” (work in progress), Feb. 2001.

[18] Oram A., (ed.): Peer-To-Peer: Harnessing the Power of Disruptive Technologies, O’Reilly & Associates, Sebastopol, U.S.A., 2001.

[19] Postel, J., (ed.) “Internet Protocol,” RFC 971, Sept. 1981.

[20] Ratnasamy, S., Karp, R., Francis, P., Handley, M., Shenker, S., “A Scalable Content-Addressable Network,” Proc. Sigcomm 2001, Aug. 2001, pp. 161-172.

[21] *Regexp* Unix Manual Pages, June 1993.

[22] Saltzer, J., Reed, D., Clark, D., “End-To-End Arguments in System Design,” ACM Transactions on Computer Systems, V2 N4, Nov. 1984, pp. 277-288.

[23] Stoica, I., Adkins, D., Zhuang, S., Shener, S., Surana, S., “Internet Indirection Infrastructure,” Proc. Sigcomm, Aug. 2002, pp. 73-86.

[24] Stoica, I., Morris, R., Karger, D., Kaashoek, M., Balakrishnan, H., “Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications,” Proc. Sigcomm 2001, Aug. 2001, pp. 149-160.

[25] Tennenhouse, D., Smith, J., Sincoskie, W., Wetherall, D., Minden, G., “A Survey of Active Network Research,” IEEE Communications Magazine, Vol. 35, No. 1, Jan.1997, pp. 80-86.

[26] Touch, J., “Dynamic Internet Overlay Deployment and Management Using the X-Bone,” Computer Networks, July 2001, pp. 117-135.

[27] Tschudin, C., Gold, R., “Network Pointers,” Proc. ACM HotNets-I, Oct. 2002, in ACM CCR, Jan. 2003, pp. 23-28.