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

Joseph D. Touch and Venkata K. Pingali

USC/Information Sciences Institute, 4676 Admiralty Way
Marina del Rey, CA 90292
{touch,pingali}@isi.edu
www.isi.edu/{touch,pingali}

Abstract. DataRouter forwards network layer packets using application layer tags, without requiring per-hop termination of transport protocols and the consequent reimplementation of transport services in the application. 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. Tagged IPv6 packets traverse non-DataRouter routers transparently. On a platform forwarding IPv4 packets at 310K packets/sec., an unoptimized FreeBSD IPv4 DataRouter 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.

1 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 [11][18]. 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.

N. Wakamiya et al. (Eds.): IWAN 2003, LNCS 2982, pp. 115-126, 2004.

1.1 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 [17].

Such services enable interesting and useful content-directed forwarding at the expense of violating the "end-to-end principle" [21]. 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 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.

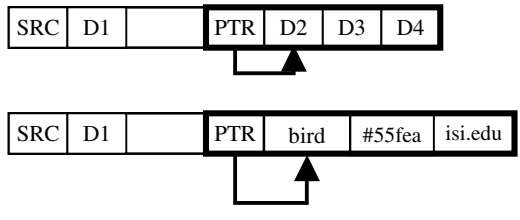

Fig. 1. IPv4 Loose Source Route (top) vs. DataRouter (bottom) options

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 [11][18]. 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 option's pointer, which is then incremented (Figure 2, left). The process stops when the pointer exceeds the option length.

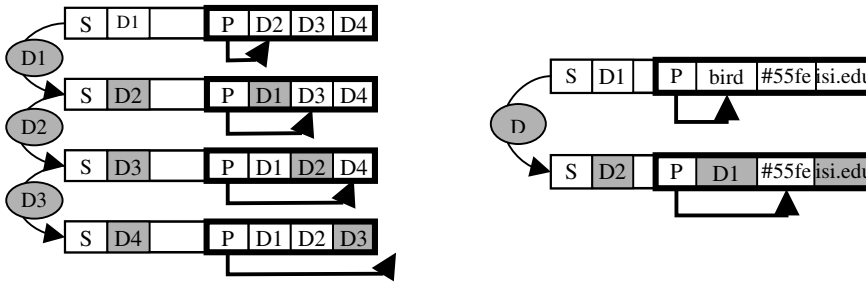

Fig. 2. IPv4 Loose Source Routing (left) and processing step in DataRouter (right)

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 2, right).

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). The key reason for per-hop connection termination is to allow the forwarder to access packet data [17]. DataRouter places that information in the IP packet header, making it accessible to the forwarder and thus avoiding the need for 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 [21]. 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][15].

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][15]. 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, but the DataRouter provides a key component of a solution.

2 DataRouter

DataRouter is a generic string match and rewriting capability at selected routers. It consists of the following components: (1) an IP option structure, (2) forwarding and rewriting tables at selected routers, (3) a fixed set of matching algorithms, (4) an API for applications to write/read the IP option, and (5) an 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.

2.1 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 [11][18]. 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 3, left).

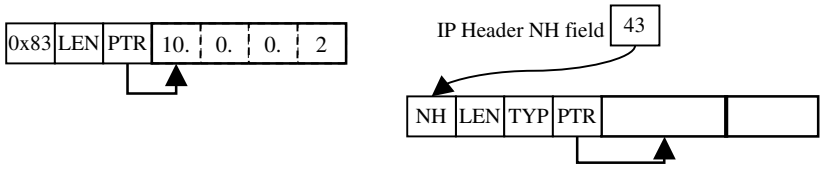

Fig. 3. IPv4 and IPv6 LSR Options

In IPv6 (Figure 3, right), 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.

DataRouter uses a chain of labeled strings rather than numeric addresses as the source route. 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 3, 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 4, 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.

The structure shown in Figure 4 is for an IPv4 DataRouter option. In IPv6, the option is a variant of the existing routing header option (Figure 4, 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.

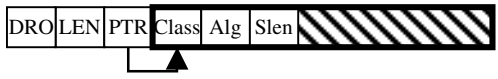

Fig. 4. DataRouter Option

The fixed set of lookup algorithms include (1) longest pattern match (2) exact match, (3) range match, (4) longest prefix/suffix, and (5) "closest" match (fuzzy match).The first three have been implemented and tested, as discussed in Section 3. The addition of a cost function (for (5)) 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. The following examples define sample forwarding methods, including a tag string, and a string-based routing class for each method:

- *DNS:* lookup=#long_sufx class=#DNS string=*joe.com*
- *URL redirection*: lookup=#exact class=#URL tag=*joe.com/apple*
- *Napster/Chord/CAN:* lookup=#exact class=#MP3 tag=*hash(song name)*
- *Google*: lookup=#closest class=#WEB string="*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 be shown in this generic scheme:

- *IPv4 LSR:* lookup=#long_prfx 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) [3].

2.2 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.
- **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.

2.3 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: (a) exact match for hashes, (b) longest prefix for IP addresses, and (c) 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: longest suffix anchored at the first single "/" – once there, remove http://www.isi.edu/ , and longest prefix thereafter. In this case, DataRouter would rewrite the current option or insert a copy before the next string to be processed. Alternately, component operations may be decomposed by the application.

2.4 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 [4].

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, providing exactly the decoupling required [16].

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

```
droute  class class_id add pat dest [alg (long|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 [20]. Further details of the pattern are under development, including how best to indicate the following: (a) match only & delete current string, (b) match & substitute on current string, and (c) if matching current string, then substitute on all subsequent strings.

The table loader API is intended to be used by either static routing commands or a dynamic routing protocol. Such a protocol could support Chord-style forwarding at the network layer, by having a Chord application insert DataRoutes into the forwarding table. The purpose of DataRouter is to support this and other kinds of forwarding in a generic service.

3 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, 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 5.

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 5 (leftmost bar). Forwarding the same packets on a kernel with DataRouter extension

support (IP/RER), i.e., data routing capability is present but not used, 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). 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. 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 results are promising, and more experimentation is planned to study the performance aspects of the DataRouter and design optimizations.

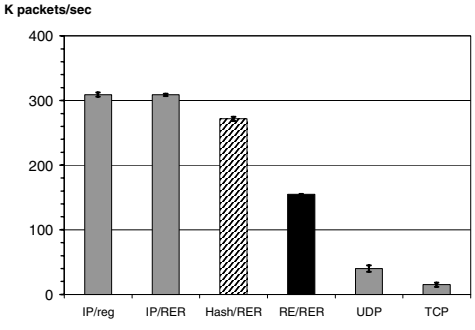

Fig. 5. Comparative IPv4 forwarding performance (in K packets/sec)

Compare these results to application layer forwarding, also shown in Figure 5 (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 [2]. 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. We suggest a technique to overcome this potential pitfall elsewhere [26].

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.

4 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 [24]. 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.

Data routing trades space within the header and slightly increased node complexity to reduce protocol complexity and application participation in network layer forwading. Trading header space for computational complexity has been explored earlier [9]. for improving route lookup performance. DataRouter uses a language similar to Data Manipulation Language [9] to encode routing instructions.

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 [19][23]. 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 cost of forwarding using packet data is large – 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 [21]. A recent approach uses data expressed as selection predicates [7]. that are constraints over a set of attribute value pairs. DataRouting supports regular expressions that are strictly more general than selection predicates. However, the constrained language of selection predicates allowed for efficient matching algorithms to be designed. DataRouter could be easily extended to incorporate the special case of selection predicates. Further, DataRouter supports string rewriting, which is not included in [7].

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][22]. 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 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 builds on virtual networking systems such as the X-Bone [25] to provide an integrated system with 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 [12]. 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 [11].

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 [6] system's tuple-style message delivery, integrated with existing IP forwarding and extended with rewriting capability.

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 TRIAD-like service [10]. Similar multilayer forwarding in Network Pointers, and Catanet can be supported [7][23][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 [5][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.

5 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. We identify several of them here but discuss only the end-to-end issues in some detail. Discussion on the rest of the issues can be found elsewhere [26].

Major issues include IPv4 transparency, optimization, late binding, and end-to-end issues. DataRouter functions can be transparently added to the network layer via a modified IP option format. Integration of the DataRouter option with existing routing protocols requires similar extensions to those protocols. The current implementation does not include any optimizations, though caching and precomputation of pattern machines may increase throughput substantially. DataRouter can support late binding, given dynamic endpoint identification negotiation, similar to emerging standards, with interesting additional requirements [13][16]. 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. Finally, the DataRouter does not require application protocols that reinvent transport services, as would violate the end-to-end principle.

As noted earlier, application layer forwarding has a negative effect on end-to-end protocols [21]. 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.

6 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.

References

1. Adjie-Winoto, W., *et al.,* "The Design and Implementation of an Intentional Naming System," Proc. ACM SOSP (OS Review, V34 N5), Dec. 1999, pp. 186-201.
2. Baker, F., "Requirements for IP Version 4 Routers," RFC1812, June 1995.
3. Borman, D., Hinden, R., Deering, S., "IPv6 Jumbograms," RFC 2675, Aug. 1999.
4. Braden, R., ed. "Requirements for Internet Hosts -- Application and Support," RFC 1123, Oct. 1989.
5. 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.
6. Carriero, N., Gelernter, D., "The S/Net's Linda Kernel," ACM Transactions on Computer Systems (TOCS), V4 N2, Nov. 1986, pp. 110-129.
7. Carzaniga, A., Wolf, A. L., "Forwarding in a Content-Based Network," Proc. Sigcomm 2003, Aug. 2003, pp. 163-174.
8. Cerf, V., "The Catanet Model for Internetworking," IEN 48, July 1978.
9. Chandranmenon, G. P., Varghese, G**.,** "Trading packet headers for packet processing," Proc.Sigcomm, Aug. 1995, pp. 162-173.
10. Cheriton, D., Gritter, M., "TRIAD: A Scalable Deployable NAT-based Internet Architecture", Stanford Computer Science Technical Report, Jan. 2000.
11. Deering, S., Hinden, R, "Internet Protocol, Version 6 (IPv6)," RFC 2460, Dec. 1998.
12. Eriksson, H., "MBone: The Multicast Backbone," Communications of the ACM, Vol.37, Aug. 1994, pp.54-60.

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. Johnson, D., Deering, S., "Reserved IPv6 Subnet Anycast Addresses," RFC 2526, March 1999.
16. Moskowitz, R., Nikander, P.,"Host Identity Payload Architecture," (work in progress), April 2003..
17. Oram A., (ed.): Peer-To-Peer: Harnessing the Power of Disruptive Technologies, O'Reilly & Associates, Sebastopol, U.S.A., 2001.
18. Postel, J., (ed.) "Internet Protocol," RFC 971, Sept. 1981.
19. Ratnasamy, S., Karp, R., Francis, P., Handley, M., Shenker, S., "A Scalable Content-Addressable Network," Proc. Sigcomm 2001, Aug. 2001, pp. 161-172.
20. *Regexp* Unix Manual Pages, June 1993.
21. 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.
22. Stoica, I., Adkins, D., Zhuang, S., Shener, S., Surana, S., "Internet Indirection Infrastructure," Proc. Sigcomm, Aug. 2002, pp. 73-86.
23. Stoica, I., *et al.*, "Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications," Proc. Sigcomm, Aug. 2001, pp. 149-160.
24. Tennenhouse, D., Smith, J., Sincoskie, W., Wetherall, D., Minden, G., "A Survey of Active Network Research," IEEE Comm. Magazine, Vol. 35, No. 1, Jan.1997, pp. 80-86.
25. Touch, J., "Dynamic Internet Overlay Deployment and Management Using the X-Bone," Computer Networks, July 2001, pp. 117-135.
26. Touch, J., Pingali, V., "DataRouter: A Network-Layer Service for Application-Layer Forwarding," ISI Technical Report ISI-TR-2003-578, May 2003.
27. Tschudin, C., Gold, R., "Network Pointers," Proc. ACM HotNets-I, Oct. 2002, in ACM CCR, Jan. 2003, pp. 23-28.