Requirements for Routing in the Application Layer

Pamela Zave

AT&T Laboratories—Research, Florham Park, New Jersey USA
pamela@research.att.com

Abstract. In the application layer of networks, many application servers are middleboxes in the paths of messages from source to destination. Applications require, as a basic coordination mechanism, a way to route messages through the proper servers. This paper elaborates and justifies the requirements for such a coordination mechanism. It presents what is known about satisfying these requirements, and what questions still need to be answered.

1 Routing as a coordination mechanism

In any networked application, routing is a fundamental execution mechanism. When a node sends a message, routing determines which node will receive it.

The most familiar form of routing is routing for the network layer of the protocol stack, especially routing according to the "classic" Internet architecture. From this viewpoint, the sole purpose of routing is to get a message to its destination.

The network literature typically distinguishes between *routing*, meaning the process by which routes are advertised and local routing tables are maintained, and *forwarding*, meaning the step in which a router receives a message, looks its destination up in a table, and sends the message out again. Because this paper is concerned more with requirements than with mechanisms, there is no need to distinguish the two concepts. Both routing and forwarding are lumped together as "routing."

The main point of this paper is that applications require routing to serve a purpose in addition to getting a message to its destination. Application servers are often middleboxes that can only do their jobs if messages pass through them on their way from source to destination. Consequently, there should be an application-layer concept of routing whose purpose is to include appropriate application servers in the paths of messages, as well as to get them to their destinations. This form of routing *would serve as a coarse-grained coordination mechanism*, because it would govern both the inclusion and order of application servers in message paths.

Like most network concepts, routing can be hierarchical. Figure 1 shows how application routing would fit into the hierarchy. In the application layer, a message M_a passes through a middlebox on the way to its destination. The

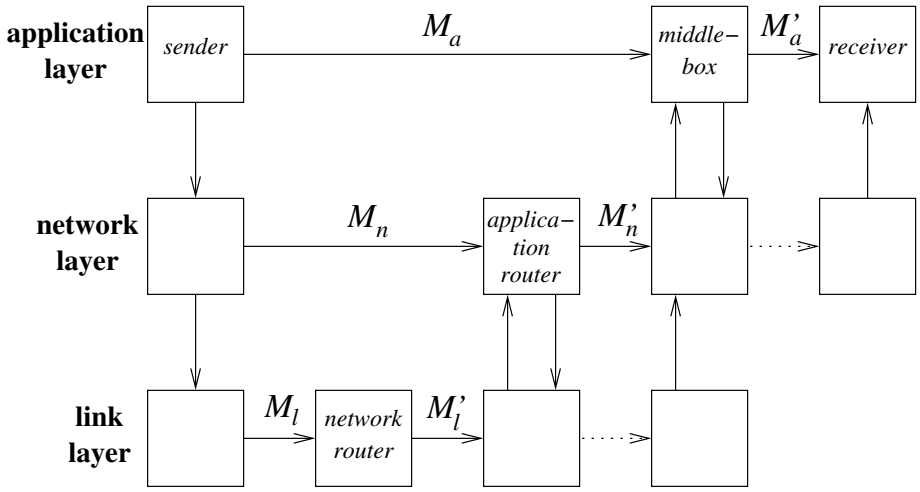

Fig. 1. Routing can be hierarchical, with application and network routing implemented in different layers. (The transport layer usually comes between the application and network layers in the protocol stack. It is omitted here because transport protocols interact little with routing.)

middlebox is a general application server, so it can modify the message to M'_a, absorb it, delay it, or replicate it.

In the network layer, M_a is encapsulated inside a message M_n that is routed to an application router by ordinary destination routing. The application router determines that M_a should be sent first to the middlebox. It changes M_n to M'_n before forwarding, so that ordinary destination routing will take it to the middlebox. A similar process goes on in the link layer, where network (IP) routers implement ordinary destination routing for the benefit of the network layer.

To set the stage for a discussion of exactly what application-layer routing should do, Section 2 gives four general reasons why an application server might be used as a middlebox rather than an endpoint. This establishes the importance of support for middleboxes.

Section 3 focuses on source/destination symmetry. This is the biggest difference between routing in the application and network layers, and hence the biggest unmet need in the application layer. Section 3 illustrates the bad effects of current deficiencies on service deployment, service maintenance, and security.

Section 4 introduces the routing capabilities of the Distributed Feature Composition (DFC) architecture [5]. DFC is a modular architecture for telecommunication services. It has been used successfully to build many voice-over-IP services, including corporate and consumer services in daily use [3, 4]. DFC incorporates application routing in a way that is appropriate to telecommunications, and provides a good example of what application routing can offer to service developers.

DFC routing has been accepted by the voice-over-IP industry, where it is part of a new standard [6] for programming application servers.

DFC routing may be a good start, but it is not sufficient to meet the needs of all Internet applications. Section 5 discusses additional requirements for routing in the application layer. It seems desirable to consider building middleware that can be shared among applications, but further research is necessary to design a sufficiently general capability.

Although this paper is focused on recommendations for the application layer of the Internet, it contains many examples from lower layers, particularly the network and link layers. This is because some of the application-layer concerns presented here are also relevant to lower layers, as described in [1] and [13]. Today's Internet has many deviations from the "classic" architecture, in which messages move transparently between endpoints. For example, a pragmatic definition of reachability in the Internet [14] combines the effects of routing, message filtering (primarily by firewalls), and Network Address Translators (NATs).

Despite the common themes found in all network layers, it seems best to focus on the application layer, for two reasons: (1) From a technical perspective, the arguments for enhanced routing are strongest and least controversial when applied to the application layer. (2) From a pragmatic perspective, the proposed application middleware would meet urgent needs of application builders, and incur relatively few obstacles to deployment. In lower layers, the need for change is not so obvious, and the obstacles to deployment are far greater.

2 Middleboxes in the application layer

In the application layer, as we would expect, some applications are provided by servers that act as the endpoints of message paths. Most Web servers function as endpoints.

The salient characteristic of the application layer, however, is that many applications are provided by servers that are not the endpoints of message paths. The view that servers can be middleboxes as well as endpoints is still somewhat controversial, because of the lasting influence of the classic Internet architecture. For this reason, it seems worthwhile to present in detail the motivations for servers as middleboxes. There are four such motivations, each discussed in one of the next subsections.

Arguments against middleboxes often invoke the end-to-end arguments [12], but these are really principles that distinguish between the network layer and the application layer. They say that the network layer should be minimal and highly efficient, because it is shared equally by all applications, and because it can perform few functions as well as application-aware software can. With this interpretation, there is no conflict whatsoever between the end-to-end arguments and application servers as middleboxes, because any application server is an "endpoint" with respect to the principles [13].

2.1 Being an intermediary IS the application

Some Internet applications have the purpose of acting as an intermediary between or among communicating endpoints. These applications can only be implemented by servers on the message paths between endpoints.

Intermediary applications perform many common functions, for example:

- They enhance security by blocking unwanted messages. This is the motivation for firewalls.
- They perform transcoding, protocol conversion, reformatting, or other functions that enable heterogeneous endpoints to communicate.
- They filter or transform content for particular audiences, for example, children or the disabled.
- They build multi-point connections out of point-to-point connections, and allow the endpoints to control them by switching and conferencing.
- They improve performance in application-dependent ways. For example, they cache Web pages.
- They improve reliability in application-dependent ways. For example, they implement automatic retrying or retargeting.

One of the most interesting categories of intermediary consists of servers representing third parties in the communication [2]. These servers might perform functions desired by the endpoints, such as acting as trusted brokers. Or they might act against the interests of the endpoints, for example by billing or wiretapping.

2.2 Servers enhance endpoints

Some network applications could conceivably be implemented in endpoints, but they are not implemented in endpoints for practical reasons.

For example, it obviously makes sense to put a voicemail capability in telephones, because most home answering machines work this way. When the endpoint device is a cellphone, however, there are important advantages to putting the voicemail capability in the network rather than in the device:

- Network voicemail provides an always-available network presence for an endpoint that is often unavailable.
- Network voicemail provides a large amount of persistent storage that is always accessible from any device.
- Network voicemail can employ speech recognition, speech search, and text-to-speech generation. Handheld wireless devices do not usually have access to the resources for such capabilities.
- Network software can be updated regularly, while small consumer devices do not usually have updatable software.

If a cellphone subscribes to voicemail in the network, then its calls must go through a middlebox that detects failure and redirects a failed call to a voicemail server.

Home-network applications can integrate two or more single-media devices into a single, virtual multimedia device. It would be very difficult to implement such an application within the devices themselves.

Finally, from a different perspective, service providers may wish to offer value-added communication services to consumers. They can only do this by implementing them in network servers and including these servers in message paths. If service providers can find a market for such services, it does not really matter whether the services could, in theory, be implemented in consumer endpoints.

2.3 Name binding

In the application layer, there are many *names* serving many application-dependent purposes. An *address* is one of a node's many names, distinguished from others only by the fact that the network layer can route to it.

Applications create and use name spaces freely. Because of this, one of the most common and important functions performed by applications is to bind one name to another. The two names involved in an instance of name binding can differ in a large number of ways, for example:

- The first name can actually refer to a group of endpoints, while the second name refers to a member of the group. Similarly, the first name can be a service, while the second name is a server performing the service (as an endpoint).
- The first name can be published and long-lasting, while the second is the current network location of the endpoint. In other words, the endpoint is mobile.
- The first name can be public or anonymous or user-friendly, while the second name is private or secret or inscrutable.
- The first name is one of many roles or aliases employed by the owner of the second name.
- The first name can be global, while the second is local to a subnetwork. Similarly, the first name can be local to one subnetwork while the second is local to another subnetwork. This type of name binding is performed by NATs and by gateways between networks with different address spaces.

The kind of name binding that comes first to most peoples' minds is binding of global names by means of universally accessible lookup services such as DNS, called *lookup binding* here. Lookup binding supports location-independent names. Several additional lookup name spaces, serving different purposes within the Internet architecture, have been proposed [9, 13]. Lookup binding is also part of popular peer-to-peer applications for file sharing and communication, each of which creates a global name space for its own users.

There is another way to bind names in the application layer, termed *path binding* to distinguish it from lookup binding. The simplest form of path binding is shown in the left half of Figure 2. The first name is the address of a server, here *a*, and the sender uses it as the destination field of a message. The addressed server itself binds this first name to a second name *a'*, changes the destination address of the message to *a'*, and forwards it. Path binding is different from lookup binding because the binding server is a middlebox in the message path between sender and receiver.

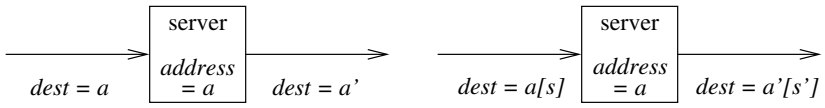

Fig. 2. *Path binding* is name binding performed by a server in the message path. The name to be bound is, or includes, the address of the server.

The right half of Figure 2 shows a simple variation on path binding in which a name consists of both an address a and a free-form string s encapsulated in the message. The address part gets the message to the binding server, while both parts contribute to the choice of second name. The properties of path binding are analyzed in [17].

Path binding is extremely common. The right half of Figure 2 depicts forwarding in a NAT, with a and a' being IP addresses, and s and s' being port numbers. If a link interface in an IP router is regarded as an implicit address, then all IP forwarding is path binding in the link layer.

Path binding is used in the application layer, in preference to lookup binding, in two situations. First, a path-binding server at address a need only bind names having a as their address part, rather than knowing how to bind all names in a namespace. Hence path binding is more local and easier to deploy than lookup binding. For example, Mobile IP [10] uses path binding to bind published mobile addresses to current network locations. This means that each "home agent" binds only its own address to the current location of its corresponding mobile endpoint, and need know nothing about other mobile endpoints.

The second situation in which applications use path binding is when they need to include an application server in a message path, and have no other mechanism with which to accomplish it. They introduce an artificial name binding for the purpose of including the server, rather than including a server for the purpose of performing a name binding. This situation is discussed further in Section 3.

2.4 Software composition

Application servers are, among other things, modules of software. *Composition*—assembling complex software by composing simpler software modules—is how we make software development "scale up."

The final motivation for using servers as middleboxes is that this provides a valuable mechanism for software composition. When one or more servers sit on a message path between two endpoints, then the relevant software of each server is composed with the software of the other servers and the software of the endpoints in a pipes-and-filters configuration. Pipes-and-filters composition makes it relatively easy to augment or change an existing application by adding to or changing the servers in message paths.

Pipes-and-filters composition is evident in the deployment of proxies and reverse proxies between clients and Web servers. It is implicit in the use of firewalls, NATs, gateways, and other common network elements. The principle purpose of DFC (Section 4) is to support pipes-and-filters composition of telecommunication features.

Multiplayer games are a rapidly-growing application area that illustrates many of the themes in this section. For such games to be playable, they must satisfy stringent requirements for scalability, latency, and fairness. Achieving this on a global scale will probably require hierarchies of middleboxes. These hierarchies will be carefully engineered to optimize performance and arbitrate fairness.

3 Source/destination symmetry

In the network layer, the sole purpose of routing is to find the destination of a message. A router can be thought of as a server working on behalf of each destination, helping to get its messages delivered. There is no need to "find" the source of the message, and there are no servers acting on behalf of the source.

In the application layer, there are also servers that work on behalf of destinations, helping their messages to find the destinations. With this important exception, most reasons for including an application server in the path of a message are potentially symmetric. If there is a destination-related reason for including a server in a message path, probably there is a corresponding source-related reason.

This is why Web terminology includes "proxies," associated with clients, and "reverse proxies," associated with servers. In the world of Web technology, there are reverse proxies for security (firewalls) and performance (load balancing). There are proxies for security (anonymizing, blocking access to some sites) and performance (caching). There are also client-side (source-related) proxy functions that have no destination counterparts; these include reformatting for special devices and filtering out annoying forms of advertisement.

In telecommunications, which is a peer-to-peer service, symmetry is even more prominent. Once a telephone call has been established, its two parties are equal. Either party (or both parties) may wish to have features that perform switching, transfering, recording, or other mid-call functions.

Source/destination symmetry is the biggest difference between routing in the application and network layers. Because the network layer has little need for source-related servers, there is poor support for them in any layer.

A possible mechanism for including application servers on behalf of the source is *source-subscription routing*. In source-subscription routing, an address *a* can *source-subscribe* to another address *a'*. If there is such a subscription, then a message with source address *a* is first routed to the node at address *a'*. If this node is a server and chooses to forward the message, then the forwarded message is next routed to its destination address in the ordinary way. Obviously, an implementation of source-subscription routing requires a bit of history in the

message, to distinguish the hop originating at the sender from the hop originating at the server.

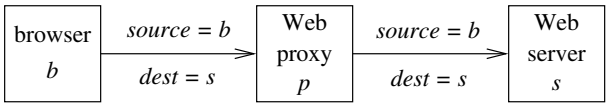

Fig. 3. Including a Web proxy in the path of an HTTP request, by means of source-subscription routing. Address b source-subscribes to address p.

It is common for network administrators to want all browsers in their subnetworks to make requests through a particular Web proxy. Figure 3 shows the use of source-subscription routing to meet this goal. Address b of the browser *source-subscribes* to address p of the proxy, which means that every message with $source = b$ is routed to p before it is routed to its destination. Note that no address translation is required to deliver the message to the Web server through the proxy.

In the absence of source-subscription routing, the common solution to the proxy problem is for the browser to send its original HTTP request with $destination = p$, and s encapsulated in the message. When the proxy forwards the message, it changes the destination to s. The proxy can be regarded as performing path binding, binding name $p[s]$ to s. This is an example of using path binding to include a server in a message path rather than to perform "real" name binding, as mentioned in Section 2.3.

This common solution has two major deficiencies. The first is that every browser in the subnetwork must be configured to use p; ensuring this is well known to be a problem for administrators. The second is that the solution relies on the cooperation of the browser, which may not be forthcoming if the interests of the browser's user differ from the interests of the administrator. Source-subscription routing avoids both the configuration problem and the security problem by keeping both information and enforcement away from the browser.

As an alternative to the use of path binding to solve the proxy problem, one might think of putting a routing list or stack $[\, p,\, s\,]$ in the message. Theoretically, this is provided for by IP source routing, although source routing is disabled in most subnetworks. In any case, a routing list has the same configuration and security deficiencies as the use of path binding.

For a second example of the use of source-subscription routing, consider the problem of making a connection to a mobile endpoint, as shown in Figure 4. Address M is the published mobile address of the endpoint, and address m is its current network location. The requestor of the connection knows the endpoint only by M. Ordinary destination routing and path binding enable the request message to pass through the location server for M and be delivered to m.

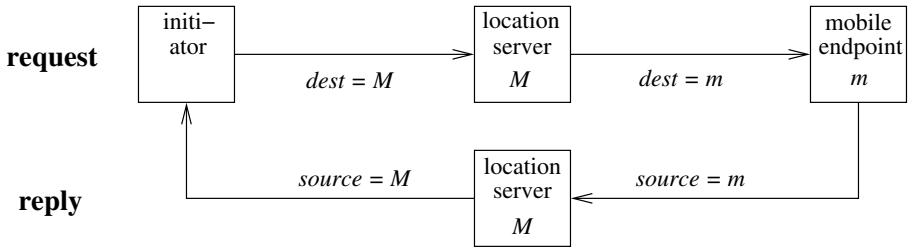

Fig. 4. Making a connection to a mobile endpoint, by means of source-subscription routing. Address m source-subscribes to address M.

The need for source-subscription routing arises when the mobile endpoint replies to the request. The correct source address for the reply is m, yet the initiator must receive the message with *source* = M. This will enable the initiator to identify the message as a reply to its request. Furthermore, the initiator must continue to send all messages within the connection with *destination* = M. This is necessary so that each message from the initiator goes through the location server and is directed to the current location of M, which may be a new address m' rather than m.

With source-subscription routing, a current location m source-subscribes to M or some other suitable server address. The source server receives the reply and changes m to M in the source field before forwarding.

This design is very similar to Mobile IP, except that Mobile IP does not have the advantage of source-subscription routing. In its absence, the mobile endpoint sends its reply with *source* = M. In most cases, M does not belong to the address space of the subnetwork of m. Hence the reply is discarded by a local firewall that performs ingress filtering. This has been the major obstacle to the deployment of Mobile IP [10].

A mobile endpoint can request a connection as well as accept a request for a connection, then move during the lifetime of the connection. In Figure 4, if the mobile endpoint were the requestor of the connection, source-subscription routing would apply to the request message. It would be routed throught the endpoint's location server, and its source address m would be changed to M.

Figure 4 has some similarities to the deployment of NATs, where M would be the single public address of the subnetwork behind the NAT, and m would be a private address within the subnetwork. In the NAT case, port numbers would be used to distinguish among the many private addresses represented by M, as in the right half of Figure 2.

The NAT case is interesting here because the reply message from m is routed through M by yet another mechanism. It is clearly not IP source routing or source-subscription routing. It is not ordinary destination routing, because M is not found in the destination field of the message. Rather, the mechanism is

an assumption that lower layers are configured so that a message *cannot* travel from m to the public Internet except by going through the NAT.

Arguably, NATs are active agents in all of the link, network, and transport layers, which accounts for the complexities that arise from them. This kind of layer spanning is neither possible nor desirable for typical applications, which may have no special administrative privileges, and which should be organized for easy maintenance. We return to the NAT example in Section 5.

4 Routing in the DFC architecture

As mentioned in previous sections, the Distributed Feature Composition (DFC) architecture supports pipes-and-filters composition of software modules. These modules are intended to implement individual, user-controllable capabilities known as *features*. In practice, the unit of composition can be a whole application server or a software module ("virtual server") within an application server. DFC uses the term *box* to encompass application servers (whether real or virtual) and endpoints.

This section summarizes the capabilities of DFC routing. The emphasis is on routing behavior and the goals it satisfies, rather than on implementation; the implementation is straightforward once the behavior is understood.

4.1 Message paths

The full path of a message is defined implicitly by the methods used in boxes. A box uses the *new* method to send a new message. A box uses the *continue* method to forward an existing message.

A full message path (see Figure 5) has a *source region* in which it is routed to boxes on behalf of the source, and a *target region* in which it is routed to boxes on behalf of the target.[1] When both source and target regions are exhausted, it is routed to its target in the ordinary way.[2]

Within the source region, boxes are included because of source-subscription routing. Within the target region, boxes are included because of target-subscription routing. Thus a message can be routed to a box because the target address subscribes to the box, then (when it is forwarded by the box) be routed to another box whose network address is the target.

The source or target subscription of an address is actually a list of boxes rather than a single box (the list can be empty). The boxes are routed to in the order listed. The boxes in a message path present because of the source (target) subscription of a single address are called the *source (target) zone* of the address.

When a box continues (forwards) a message, it can change the source or target address. If the box lies in the source region and changes the source address, then

[1] DFC terminology uses "target" instead of "destination", for brevity.

[2] It is also possible to define a *network region* between the source and target regions, for boxes that provide network functions such as billing.

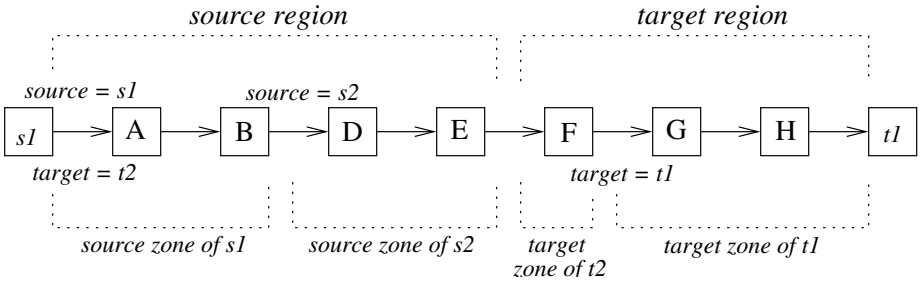

Fig. 5. A message path created by DFC routing.

any remaining boxes in the source subscription of the original source address are omitted from the message path. Rather, routing immediately begins to traverse the source subscription of the new source address. For example, in Figure 5, the source-subscription of address *s1* is [A, B, C]. Box B changes the source address to *s2*, however, so rather than going to box C, the message is next routed to D. Similarly, if a box in the target region changes the target address, then the next box in the path is the first box of the target subscription of the new target address.

We are already familiar with target-region boxes that change the target address, because this is just another form of path binding. It is also useful, however, for source-region boxes to change the source address. For example, a box could change the original source address, which is the address of the calling device, to a personal address associated with the person sending the message. This has two benefits. First, the personal address is a better identifier of the sender for the benefit of the receiver. Second, the message can then be routed through boxes subscribed to by the personal address, thus utilizing their functions.

DFC routers must have access to the relevant subscription data. The necessary routing history of an individual message is carried along in the message, so that routers do not maintain state at that level. The routing history may be encrypted so that boxes cannot read it.

4.2 Composition, modularity, and additive authority

The path mechanisms in the previous section meet the application-layer requirements of composition, modularity, and additive authority.

Composition dictates that there is no such thing as a unique server. If there is a reason why one server should be included in a message path, then the same reason might also apply to multiple servers. DFC routing supports composition in two ways. First, a subscription can be a list of any length, so that a zone can contain any number of boxes (servers). Second, a region can contain any number of zones.

Modularity dictates that servers should not know or need to know which other servers are present in the message path. When this requirement is satisfied,

each server can be developed independently. Also, server configurations can be changed easily, because the servers themselves need not change. DFC routing was designed very specifically to satisfy this requirement.[3]

The most common violation of the modularity requirement is the use of path binding to include servers in message paths (Sections 2.3 and 3). Often the purpose of these servers has nothing to do with name binding. If multiple servers are required, and if path binding is the only available means to include them, then every server must know the address of the next server in the path.

Composition and modularity can be illustrated with Web proxies. A user might wish to use all three of these proxies:

- A text-to-speech "edge service" proxy, which improves Web access for the visually impaired.
- Privoxy, which is a filtering proxy that can remove advertisements and other "Internet junk."
- A Tor client. The Tor client encrypts and chooses an anonymized, random route through participating Tor nodes to the requested server. The purpose of Tor is to protect the user's privacy, particularly from attacks by traffic analysis.

The proxies must be applied to each HTTP request in the order shown in Figure 6. Because the text-to-speech proxy gets each request before Privoxy, it gets each Web page after Privoxy has filtered it. Because Privoxy gets each request before the Tor client, it gets each Web page after Tor has decrypted it.

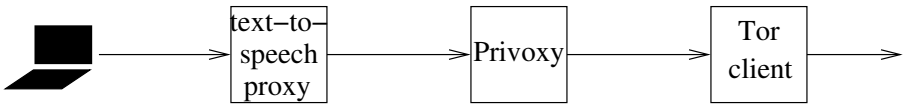

Fig. 6. Using multiple Web proxies.

With DFC routing, it would be easy to deploy the proxies in this way. It would also be easy to make Tor optional for each such user, by allowing the user to toggle his subscription to the Tor client. This is valuable because fetches through Tor are inevitably slow.

With current technology, on the other hand, each proxy must be configured to send requests to the next one. It is impossible to remove Tor from the message path without altering the configuration of Privoxy. If Privoxy is running as a shared server in a local-area network, then it is impossible to apply Tor to the requests of some users and omit it from the simultaneous requests of other users.[4]

[3] This simplified view of modularity is appropriate for discussion of routing, which provides coarse-grained coordination. Finer-grained coordination may require analysis and management at the protocol level, as exemplified by [15].

[4] I am indebted to Trevor Jim for this example.

Finally, *additive authority* means that a server should be included in a message path if one of the endpoints desires it *or* if the relevant administrator desires it. As a general security requirement, it should be easy for an administrator to ensure that all messages of a particular type go through a particular security server. DFC routing supports additive authority because our implementations allow both parties to contribute to the subscription lists. In practice, almost every subscription has contributions from both parties.

4.3 Usages

DFC is connection-oriented. For this reason, the only messages that are routed as described above are the messages that request connections. Within each inter-box hop of a request message, the reply message forms an independent box-to-box connection. All subsequent signaling takes place within these connections.

DFC boxes and signaling connections form dynamic graphs called *usages*. DFC usages have many interesting properties and behaviors, some of which are illustrated by the usage in Figure 7.

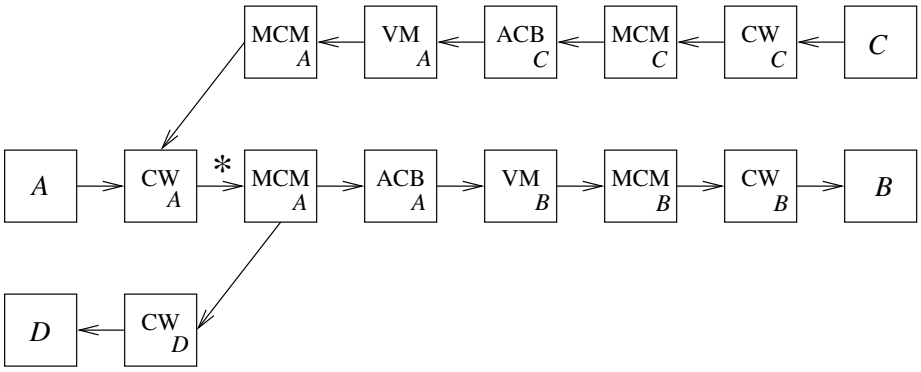

Fig. 7. DFC usages are dynamic graphs that evolve over time.

There are four user endpoints: *A, B, C,* and *D*. DFC subscriptions actually name *types* of boxes rather than boxes themselves. Each of the four endpoints subscribes to the same box types, namely [CW, MCM, ACB] in the source region, and [VM, MCM, CW] in the target region. The acronyms stand for Call Waiting, Mid-Call Move, Automatic CallBack, and Voice Mail, respectively. Voice Mail is a target-side failure treatment: if the endpoint cannot be reached, redirect the call to a voicemail server. Automatic CallBack is a source-side failure treatment: if an attempt to place a call fails, remember it and try again later.

DFC maps between box types and boxes as follows. There are two categories of box type, *free* and *bound*. When a DFC router needs to route to a box of a free type, it simply creates a fresh new instance of the box type. In contrast, there

is only one instance of a bound box type for each address that subscribes to the type. When a DFC router needs to route to a box of a bound type, it routes to the unique instance associated with the relevant address. In this example, Call Waiting is a bound type, and all others are free.

In Figure 7, A has placed a successful call to B. Each box instance is labeled with the address on behalf of which it is included in the usage. The arrows show the direction that the request message traveled to set up these connections.

In Figure 7, C has also placed a call to A. This call is also successful, because A has Call Waiting, which accepts the call. In this example, CW and MCM are *reversible* box types, which means that a subscriber subscribes to them in both regions, because they are useful to both callers and callees. Reversible box types are the ultimate example of source/destination symmetry. Although the request from A to B was routed to CW_A in the source region, the request from C to A is routed to CW_A in the target region. Now A has a signaling connection to both B and C, and can switch the voice channel so as to talk to either B or C at any moment.

A represents a user's home phone, and D represents the same user's cellphone. While A was connected to C and talking to B, the user noticed that it was time to go to work. He invoked the function of Mid-Call Move to move the B conversation to his cellphone, resulting in the third row of the figure. In the next instant the Mid-Call Move box will drop the connection marked with an asterisk, resulting in two separate usages. The user can hand telephone A to another family member to continue talking to C. Alternatively, hanging up telephone A will tear down the entire usage to C.

This is a specific example of a general pattern, which is that a long-lasting connection can be made up of connection segments that were not set up at the same time or in the same direction. The connection from A to C has segments set up from A to CW_A and from C to CW_A. The connection from B to D has segments set up from MCM_A to B and from MCM_A to D. Routing correctly in this situation requires a third method: in addition to forwarding with the *continue* method, a box can forward with the *reverse* method [16].

Similar structures might also arise in automotive infotronics, in which most communication consists of streams of sensor/actuator data. A sensor should be engaged in *some* end-to-end connection at all times, to ensure that critical data is not being lost, which means that some connection segments will be very long-lasting. At the same time, the exact configuration of data streams and functional components will vary with the vehicle's current mode of operation.

Despite the subtlety of usages, it is possible to prove useful properties about them [16]. For example, *any* connection between two endpoints X and Y with these same subscriptions contains the following subsequence of box instances: [CW_X, MCM_X, MCM_Y, CW_Y].[5] This invariant is true regardless of how the connection was initially formed or how it has evolved over time.

[5] After the instance of MCM_A moves the connection to D, we consider it an instance of MCM_D. It can be signaled from D to move the connection again.

5 Unmet requirements and future work

This section considers the possibility of building middleware that performs routing for the application layer. It would be deployed in the network as shown in Figure 1. It would make applications easier to build, deploy, maintain, and improve. It would also make them more secure.

This middleware should be shared among applications, amortizing its cost and, more importantly, facilitating convergence among applications. The value of current Internet applications is greatly reduced by the fact that each one tends to be an isolated fiefdom. Furthermore, this isolation limits imagination and innovation, as it prevents us from seeing the potential relationships among applications.

5.1 Deficiencies of DFC routing

It seems that the DFC routing capability is a good start toward meeting the routing needs of Internet applications, but it is not sufficient. This section discusses its deficiencies for this purpose, and the research needed to remove the deficiencies.

DFC routing was designed for the single application of telecommunications. It has been amply illustrated that the principles of DFC routing are relevant for other applications, taken individually. Now we need to understand how routing could contribute to application convergence, which might mean routing messages through servers associated with different applications.

DFC routing was also designed for a single administrative domain. Multiple administrative domains are a fact of life in the Internet, and there are many questions about how DFC routing should be extended to include them. For example, consider the source and target regions in Figure 5. If the message path extended across multiple administrative domains, would it still look like this, or would there be a source and target region of the message path for each domain? Interestingly, there are good arguments for both answers to this question.

All the other deficiencies of DFC routing come down to efficiency in one way or another. As noted in Section 4, DFC routing is most often used to route messages to virtual servers within a physical application server. In this context, efficiency is not critical, and has received correspondingly little attention.

First, DFC routing often routes to boxes that will never be activated in this particular communication. These boxes simply behave transparently throughout their lifetime. The inefficiency of including transparent servers in message paths could be reduced by offering finer-grained selection criteria.

For an example, let us return to the NAT example at the end of Section 3. Say that a node with private address m is replying to a connection request from the open Internet. We want to use DFC routing to route the reply through a NAT, which will change the source address from private m to public M. Source-subscription routing is not very good for this purpose, because it will route *every* message from m to the NAT. If the message is destined for another node within the same subnetwork, then the NAT must behave transparently. An optimal

subscription mechanism would route the message to the NAT only if the source is local to the subnetwork *and* the destination is not.

Second, a DFC request message goes through a DFC router each time it is forwarded. There should be a way to make the router visit optional, for situations in which it is not required. Note, however, that the mandatory router visit is an important security mechanism. Any optimization must be carefully designed to leave the security intact, allowing only trusted parties to circumvent it.

Finally, all DFC messages other than requests retrace the exact paths laid down by requests, with no servers omitted. Yet there are many situations in which subsequent messages could take a more direct route. The Session Initiation Protocol [11] allows the reply to a request to skip servers by distinguishing between *record-route* proxies and others. Even with DFC-style signaling to control IP media channels, the media packets themselves can travel directly from end to end [18].

5.2 Related work

In the current Internet architecture, the only mechanisms available for influencing or altering ordinary destination routing are IP source routing, underlying network topology, path binding, and lookup binding. None of these as currently used is secure, robust, and general enough to meet the full spectrum of application requirements.

In [1], lookup binding is used to achieve the effective equivalent of destination subscription routing in DFC: a sender looks up the destination (a global, location-independent identifier), yielding a route to traverse to get to the destination. Section 2.3 showed that lookup binding and path binding are architecturally different ways to bind names. Section 1 showed application-level routing as implemented by path binding at a lower level. The use of lookup binding in [1] shows us that lookup binding is another implementation possibility for application-level routing.

There are several well-known middleware systems to support each of the publish/subscribe architecture and grid computing. There are also middleware systems for other coordination architectures such as distributed tuple spaces, e.g., [8]. In general these architectures are not concerned with middleboxes in message paths. As a result, their routing activities tend to be overlays on ordinary destination routing. For example, a grid architecture can be used to locate a desired resource for a potential client of that resource. The location is a destination address, which the client proceeds to use in the ordinary way.

Service-oriented architecture is such an active area that there are several service-oriented architectures. At the simpler end, as above, the architecture helps to locate a service for a client. Once located, the service is reached by ordinary destination routing. At the more complex end, "choreography" languages such as WS-CDL are used to create global descriptions of distributed Web-based services. As this technology matures, we will see whether choreography is compatible with DFC routing, or whether it requires tight logical coupling between

servers. In the latter case, the compositional freedom supported by DFC routing would not be required or even allowed.

A new effort to formalize what routers (in the most general sense of the term) do promises to be relevant to the application layer [7]. In particular, it can help to define the space from which potential optimizations can be chosen.

5.3 Future work

Before we can propose a specific middleware system for routing in the application layer, three broad open questions must be answered.

First, the most powerful routing functions are expensive in terms of visits to servers and routers. We must understand how to provide a large range of cost/function trade-offs, and how to guide users in making choices.

Second, the proposed new middleware must support convergence of different applications, and it must compose well with middleware for other purposes. These are areas in which there is little general knowledge, and much research to be done.

Third, we must understand how to extend the current routing scheme to multiple administrative domains. Both routing and subscription mechanisms must be examined from a security perspective.

6 Summary

This paper has explained why routing through middleboxes is an important coordination mechanism in the application layer, and justified the claim with examples from Web services, home networks, telecommunications, mobile IP, automotive infotronics, and multiplayer games.

The current mechanisms available for influencing routing in the Internet are not general enough to meet the needs of applications, nor do they enhance security or facilitate the deployment and maintenance of applications. The biggest gap is caused by the fact that applications have a great deal of source/destination symmetry, while routing at the network level is focused exclusively on the destination.

The application-specific routing capability of the DFC architecture is a better model, and meets many requirements that are shared by all applications. However, the fact that it was designed for a specific application means, inevitably, that some requirements of different applications are neglected. This paper identifies the deficiencies and explains the questions that must be answered before the routing requirements of all applications can be met.

References

1. H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A layered naming architecture for the Internet. In *Proceedings of SIGCOMM '04*. ACM, August 2004.

2. M. S. Blumenthal and D. D. Clark. Rethinking the design of the internet: The end-to-end arguments vs. the brave new world. *ACM Transactions on Internet Technology*, 1(1):70–109, August 2001.
3. G. W. Bond, E. Cheung, H. H. Goguen, K. J. Hanson, D. Henderson, G. M. Karam, K. H. Purdy, T. M. Smith, and P. Zave. Experience with component-based development of a telecommunication service. In *Proceedings of the Eighth International Symposium on Component-Based Software Engineering*, pages 298–305. Springer-Verlag LNCS 3489, May 2005.
4. G. W. Bond, E. Cheung, K. H. Purdy, P. Zave, and J. C. Ramming. An open architecture for next-generation telecommunication services. *ACM Transactions on Internet Technology*, 4(1):83–123, February 2004.
5. M. Jackson and P. Zave. Distributed Feature Composition: A virtual architecture for telecommunications services. *IEEE Transactions on Software Engineering*, 24(10):831–847, October 1998.
6. JSR 289: SIP Servlet API Version 1.1. Java Community Process Early Draft Review, `http:// www.jcp.org/ en/ jsr/ detail?id=289`, 2007.
7. M. Karsten, S. Keshav, and S. Prasad. An axiomatic basis for communication. In *Proceedings of the Fifth Workshop on Hot Topics in Networks*. ACM SIGCOMM, 2006.
8. A. L. Murphy, G. P. Picco, and G.-C. Roman. Lime: A coordination model and middleware supporting mobility of hosts and agents. *ACM Transactions on Software Engineering and Methodology*, 15(3):279–328, July 2006.
9. M. J. O'Donnell. Separate handles from names on the Internet. *Communications of the ACM*, 48(12):79–83, December 2005.
10. C. E. Perkins. Mobile IP. *IEEE Communications*, May 1997.
11. J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler. SIP: Session Initiation Protocol. IETF Network Working Group Request for Comments 3261, 2002.
12. J. Saltzer, D. Reed, and D. D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 2(4):277–288, November 1984.
13. M. Walfish, J. Stribling, M. Krohn, H. Balakrishnan, R. Morris, and S. Shenker. Middleboxes no longer considered harmful. In *Proceedings of the Sixth Usenix Symposium on Operating Systems Design and Implementation*. ACM, December 2004.
14. G. Xie, J. Zhan, D. A. Maltz, H. Zhang, A. Greenberg, G. Hjalmtysson, and J. Rexford. On static reachability analysis of IP networks. In *Proceedings of IEEE Infocom*. IEEE, March 2005.
15. P. Zave. An experiment in feature engineering. In A. McIver and C. Morgan, editors, *Programming Methodology*, pages 353–377. Springer-Verlag, 2003.
16. P. Zave. Ideal connection paths in DFC. Technical report, AT&T Research, November 2003.
17. P. Zave. Compositional binding in network domains. In *Proceedings of the Fourteenth International Symposium on Formal Methods*, pages 332–347. Springer-Verlag LNCS 4085, 2006.
18. P. Zave and E. Cheung. Compositional control of IP media. In *Proceedings of the Second Conference on Future Networking Technologies*. ACM SIGCOMM, 2006.