Internet Engineering Task Force
Internet-Draft
Expires: May 2001

V Guruprasad
IBM T J Watson Research Center, NY 10598

November 2000

The Case for an Addressless Internet
draft-guruprasad-addressless-internet-00.txt

Status of this Memo

Abstract

This Internet-Draft proposes elimination of end-to-end IP addresses from the Internet, presenting a stronger embodiment of the end-to-end principle by elevating the end-to-end aspects of connectivity and routing above the network to an operating system-like framework that implements virtual network addressing and efficient translation to switched paths, obviating end-to-end addressing at all levels. The resulting flexibility would allow, for example, piecewise transition to IPv6 and full access to the existing Internet for the emerging wireless services without allocating new addresses.

Contents

1 Introduction 4
1.1 Overview 4
1.2 The fundamental problem 6

1.3 The fallacy of stateless routing 6
1.4 The OS parallel 8
1.5 Space-filling translation 8
1.6 Protocol stack 9
1.7 Similarities and differences 10

2 Architecture 11
2.1 Concept of virtual network addressing 11
2.1.1 Open growth and deployment 11
2.1.2 Offline routing and optimisation services 12
2.1.3 Security by invisibility 12
2.1.4 Unification with telephony 12
2.1.5 Domain transparency 13
2.1.6 Mobility 13
2.2 Principles of design 14
2.2.1 Reversal of roles 14
2.2.2 Service indirection 14
2.2.3 Route-interpretation of user request 15
2.2.4 Homogeneised URLs 15
2.3 Addressless connectivity 16
2.3.1 End to end connectivity 17
2.3.2 Comparison with DNS/IP 17
2.3.3 Relation to Web servers 18
2.3.4 Similarity to telephony and email 18
2.3.5 Locality in control plane 18
2.3.6 Open extensibility 19
2.3.7 Efficiency of routing 19
2.3.8 Stateless interpretation 20
2.3.9 Control plane optimisations 20
2.3.10 ACS shortcuts and forward-encoding 20
2.3.11 Intelligence and admission control 22
2.3.12 Security and resilience 22
2.3.13 Prediction database 23
2.3.14 Optimisation by k-locality 23
2.4 Addressless transport 24
2.4.1 Two-way path stitching 24
2.4.2 Delayed signalling and asymmetry 27
2.4.3 Stateless signalling 28
2.4.4 Inclusion of network state 28
2.4.5 QoS attributes and alternative paths 29
2.4.6 Forward encoding with translation 29
2.4.7 Parallel paths 30
2.4.8 Locality of network information 30
2.4.9 Passive and active aggregation 32
2.4.10 Transport security 32
2.5 Universal user interface 33
2.5.1 Distributed parallel support 33

2.5.2 Interface route tables 33
2.5.3 IPC and DFS-like abstractions 34
2.5.4 Modified API arguments 34
2.5.5 Sample system calls 35

3 Summary 36
3.1 Features and performance 36
3.2 Deployment and coexistence 36
3.3 Open extensibility 37
3.4 Administration 37
3.5 Security 37

4 Intellectual property 38

List of Figures

1 Architecture overview 5
2 Equivalence of routing state 7
3 Space-filling translation 8
4 Protocol stack 9
5 Control and transport planes 16
6 Peephole optimisation 23
7 Overlapping shortcuts 23
8 Stitching algorithm: request pass 25
9 Stitching algorithm: acknowledgement pass 26

1 Introduction

1.1 Overview

The fundamental problem of networking, which had hitherto necessitated end-to-end (e2e) addressing, lies in the laziness permitted to server applications, in that they wait at their own hosts, forcing unique identification and e2e routing all the way from the client end. In addition, end-to-end (e2e) addressing violates the e2e principle by

- requiring e2e network knowledge (omniscience) at each node in the form of e2e routing tables (Section 1.3);
- letting network size, instead of volume, affect local traffic by imposing larger addresses ("cosmology syndrome", Section 1.3);
- posing global consistency and conformance constraints that hinder not only growth but even the migration to IPv6 (homogeneity);
- and limiting the network size without allowing the host system and application software a chance to do better (bound).

The fundamental problem is solved, as elsewhere described [1], by

- treating connectivity and e2e routing as application issues,
- providing a distributed virtual space of services over which server and client requests can meet very expeditiously half-way by simply interpreting URLs over this space,
- and efficient translation of the request traversals to virtual paths, exploiting the spanning tree property of the virtual space and attribute grammars to represent network information.

The framework adapts the paradigms of virtual addressing (Section 1.4) and program translation (Section 1.5) from host system software to the network domain, and by moving e2e connectivity and routing out of the network, takes the e2e principle further and becomes deployable over and independently of existing networks and the Intranet. Only local network information is involved in setting up the virtual space and in the translation, avoiding network omniscience as well as permanently relaxing the homogeneity constraints, so that even the transport plane becomes more readily extensible. Virtual paths over addressing network islands and across switches eliminate e2e addresses and the size bound altogether, and provide scalability through hierarchical tunnelling without recreating the "cosmology syndrome".

The translation efficiency is linear in the path length, instead of depending on network size, as in both IP with CIDR and ATM PNNI with either source or hop-by-hop routing, but depends on a single spanning

tree. However, the tree is populated by the applications themselves and is highly extensible, encouraging rapid lateral growth, and the neighbouring network spanning trees can be readily incorporated into the translation, so that the growth would be "space filling" (Section 1.5) and the routing, opportunistically complete and optimal.

As depicted in Fig. 1 below, the framework is proposed not as a replacement for existing networks or the Internet, but as a separate control plane with overlaid switches, hosting the virtual space of services and providing an overlay of virtual paths across both switched and addressing networks, the latter including the existing Internet. In particular, the framework allows use of IP to address both data streams to cellular devices (Section 2.1.4) and exploitation of the existing Internet by enabled applications that would be additionally able to transcend the address bound (Section 2.1.5). Additionally, the framework provides an enhanced degree of security

[This contrasts with the IPv6 proposal, for example, which imposes changes in host application software, without substantial immediate benefits to justify the migration effort, and in the infrastructure, where the improvements are not envisaged to be lasting nor quite as substantial [2], and is certainly not a formal, permanent solution of the fundamental problem of networking.]

```
     /~~~~~~~~~~\___/~~~~~~~~~~~-~~~~~\____/~~~~~~~~~\           ---
    /  *   *             *        **          *  *    \           ^
   (     Virtual address space of services (URLs)   *  )          |
    \ *       *   * *          * *         *      *   /        control
     \___/\____________/\____________/---------\_____/           plane
                              |          |                        |
                              |          |                        |
    ---                 --v--        | routing translation   -x-
  -------  virtual paths  ----v----      v                        |
 ====(+)==========(+)========(+)======(x)======(x)=======(x)=  overlay
      |            |          |        |        |         |       |
      |            |   -- network plane --     |         |       v
 /---------\  /---------\  /-----\  /-----\  /------\  /------\  ---
 | v4 Inet |  | v6 Inet |  | VPN |  | ATM |  | MPLS |  | UMTS |
 \---------/  \---------/  \-----/  \-----/  \------/  \------/

      addressing islands <----           -----> switched
```

Fig. 1: Architecture overview

1.2 The fundamental problem

As stated above in Section 1.1, the fundamental problem is the traditional application programming interface (API) notion of client applications having to reach all the way to the application server hosts. All other issues, including the e2e principle, appear to have been secondary in this regard. It is the main reason why ATM switches and interfaces have needed unique addresses, and why MPLS, which does not envision e2e addresses of its own, needs to rely on IP.

Solving this problem would have entailed touching the API, which had been viewed as a subject outside the networking domain. However, the e2e addressing solution is so simplistic that application programs have had to deal with the network addresses, as shown by the sockets API [3]. Even if better dressed as IP addresses, network addresses are real addresses in that they locate physical destinations in the network, unlike memory addresses, which are routinely virtualised by host operating systems.

On the other hand, with the network willing to take on this load, there had been little motivation in the host software space to examine connectivity and route management roles. Instead, even system software like distributed file systems (NFS, DFS), interprocess communication (DIPC) and parallel processing libraries (PVM, MPI) invariably depend on the "network" for addressing, connectivity and routing.

An analogous operating system layer for the network, providing network address virtualisation ought to be more application friendly, and this is ensured in the present framework by the use of URLs and automatic translation. The API impact is very slight and entirely favourable, appearing as a extension of the popular Web paradigm (Section 2.5.4). Security is enhanced by capability for server invisibility and pre-connect authentication, and by connection-oriented data transport (Section 3.5).

1.3 The fallacy of stateless routing

IP is traditionally claimed to have "stateless" routing (cf. [4, Section 2.2.3]), but the claim is in fact fallacious, as it overlooks a fundamental equivalence between the claimed state and the network state information embodied in IP router tables.

Specifically, the claim fails to take into account the facts that for scalability, any routing paradigm would eventually employ hierarchy, and that the higher levels of the hierarchy would be automatically expected to be more static and less branched, and it is the latter

that constitute the bulk of the in-network state. The label stack of MPLS illustrates this correspondence as shown in Fig. 2, with the pre-CIDR IP address classification scheme, as follows.

```
        ------------------------------
        | L3 | L2 | L1 | L0 | payload |              switched packet
        ------------------------------
           |                \
           |                 \
  ---------------------------------------------------
  | ... | AAAA | BBBB | CCCC | DDDD | ... | payload |   IPv4 packet
  ---------------------------------------------------
        |<--- IPv4 address bits --->|
```

Fig. 2: Equivalence of routing state

The outer labels refers to network topology state, since the virtual path tunnels are relatively static and form long haul WAN networks, and are independent of the e2e connections. The higher order bits of IP addresses similarly represent long haul WAN routes, which are not only analogous to the path tunnels, but are often literally routed over those very tunnels. The outer labels and the higher order IP address bits thus both represent network topology state, invalidating the "statelessness" claim. Incidentally, IP is by definition less scalable, because only one layer of IP tunnelling gets to fulfill the notion of e2e addressing. More significantly, because of CIDR, IP increasingly suffers from the problem of omniscience at each router node, which is not the case with virtual paths, particularly under the present architecture.

IP also suffers from the "cosmological effect", viz that all traffic, even between neighbouring nodes, is forced to carry more addressing bits as the network increases, albeit in logarithmic blocks, whereas in the virtual path hierarchy, the penalty of additional labels on the link bandwidths and latencies would be incurred, in the same block-logarithmic fashion, in proportion to actual traffic volume in the given path and not the size of the network.

In retrospect, the only fair comparison that can be made is that the analogous notions of hierarchy have been simply less obvious and understood in the switched world, and came long after IP routing had been operating e2e. The related problem that switched paths define rigid routes seems to have known or obvious solutions, including statistical load balancing across virtual paths and numerous techniques for repair, recovery and self-healing.

1.4 The OS parallel

There is a close, obvious equivalence between NAT and bank-switching, used in embedded systems as a form of virtual memory, but is not a complete substitute for the per-process virtual address space as obtained, for example, in a modern Unix-like operating system (OS) for memory addressing, the principle limitation being that bank-switching merely remaps portions of the address space, instead of extending it.

Address chunk replacement is not a issue for individual users, but becomes a problem for services expecting a large number of devices that would occupy to occupy a sizeable chunk of the address space, as envisaged for the next generation cellular telephones (Section 2.1.6).

Both the global nature of addresses and the practical bounds on real addresses are solved problems in OS theory, albeit only in the context of host memory management, and programming languages and compilers have evolved alongside. The present architecture in effect proposes a similar evolution from primitive DNS functionality to full e2e connectivity, including integration with multipoint connectivity management hitherto handled by user space libraries like MPI, and the functionality is appropriately consolidated into an OS-like control plane conceptually below the applications, but above the network.

1.5 Space-filling translation

Space-filling arises from the combination of the lateral growth expected in the control plane, by providers and users adding to the virtual service space, and the translation, as explained below with the help of Fig. 3.

```
 \                         to A     Q ||    f       to B     ---
  o A    |        B        <---  .....()....*.....  --->      ^
 / \     o        o                    \\  . .     .          |
    \     \ P    / \                   P \\.   .   .      translation
     >-----o----<        |-->    ========()===========      swath
    /      |     \                       ||     * g         |
           o R                    .    R ()    . .          v
          / \                       .   //\\  .   .          ---
         /   \                        .//  \\.     .
                                      // . .\\
                                     // h *  \\
```

Fig. 3: Space-filling translation

The figure shows a sample region of the virtual space on the left, which constitutes a spanning tree for reasons that will become clear in Section 2.3. A neighbourhood of a node P in the tree is shown magnified on the right, the dotted links representing a second network in the same region. Translation is necessary in the present scheme because e2e connections are efficiently established first in the virtual space, but the tree would not be generally designed to support data traffic. In principle, a data network can be laid out alongside, but that would be restrictive and unoptimal.

For example, the shortest physical path between points A and B in the figure might not pass through P , and if such a physical path, say A-*-f-*-B, is or can be made available, the framework should enable its use, which can be achieved without e2e addressing, as will be explained in Section 2.4, by translating from the e2e control path A-*-P -*-B to the desired physical path, say A-*-f-*-B.

It will be shown in Section 2.4.8 that a large but finite "swath" over the transport network can be swept in the translation process, which physically follows the e2e control path. The swath together with the lateral growths expected in both the virtual space and in the supporting networks, guarantees space-filling,

1.6 Protocol stack

As stated in the overview (Section 1.1), the framework sets up transport primarily through virtual paths, relegating legacy ATM and IP networks, including the existing Internet, to the status of regional media. Consequently, the virtual path termination comprises the highest network interface layer below the flow control and application specific protocols, as shown in Fig. 4 below.

```
\         application      /
|--------------------------|
| HTTP/RTP |      other    |
|----------|     stacks    |
| TCP/UDP  |               |
|--------------------------|
| virtual path termination |
============================
| IPv4 | IPv6 | ATM | MPLS |
```

Fig. 4: Protocol stack

1.7 Similarities and differences

The notion of URLs is now well established by the World Wide Web. The URLs are homogeneised in the present approach, as there can be no inherent notion of physical network addresses at the application level, and look more like a pure filesystem pathname (Section 2.3.1). More significantly, URLs are used for all e2e connectivity, i.e. for all networking activity other than network configuration and administration (Section 3.4), whereas URL usage are currently confined to a small family of HTTP and related protocols in the Internet.

The Web itself provides numerous virtual registeries of services, but all such registeries are currently "hosted" or "homed" at real network locations identified by IP addresses. For example, the Universal Description, Discovery and Integration (UDDI) specifications define format and methodology that would be used by a number of business-to-business (b2b) portals rather than a single universal service space like the DNS.

Hierarchical nesting of switched packet labels is specified in MPLS for the same reason of scalability but MPLS does not address e2e connectivity and is currently dependent on IP for this purpose. The present analysis also shows that the MPLS label stack hierarchy in fact addresses the issue of state more thoroughly than IP with CIDR (Section 1.3).

While the present architecture was partly inspired by the peer-to-peer messaging notions in MPI [5], and provides an elegant unification of MPI messaging with traditional point-to-point network connectivity, the unification is purely conceptual. There are many different ways available for distributed applications, as well as a multitude of issues relating to them; all that has been ensured is capability for supporting them using a single API.

2 Architecture

2.1 Concept of virtual network addressing

In the strict sense, virtual network addressing would mean that applications (and users) no longer have to deal with network addresses, which are real in the sense of identifying physical locations within the network.

It does not mean that network addresses cease to exist altogether, but rather that their role would be confined to a lower level. It also does not mean their complete invisibility, because a translation to real addresses is always necessary. Two constraints are important in the networking context:

- only real addresses should be used for data transport, and
- any form of address directly connecting application processes would not be virtual.

Both are trivially obeyed in IP because the same address is used for both connectivity and transport roles. The second constraint means that virtual addressing is impossible in the absolute sense, and the best that can be had is indirection.

It will become clear from Section 2.3, however, that even in the limit of directness, e2e connectivity can be efficiently obtained, with all the benefits of virtual addressing, described below, so long as the connectivity is obtained by en route translation from the application space to non-e2e real addressing within the network layer.

2.1.1 Open growth and deployment

The most important benefit of virtual addressing, or addresslessness in the e2e sense, is the elimination of the current constraint of global address space consistency on the addition of new services and client devices to the Internet. This should allow the Internet to advance more rapidly as the administrative infrastructure for address management would be reduced or eliminated.

Furthermore, the above-network nature of virtual addressing means that with well-defined interfaces to networking and host software, the virtual addressing layer can be provided by independent entities. It would be even possible for these virtualisation providers to share the networks or host customers.

2.1.2 Offline routing and optimisation services

Another provider opportunity lies in the network topology area. Since e2e knowledge of the network is intentionally not assured in the translation process, routing cannot be guaranteed in all cases, nor to be optimal in a global sense.

The Internet itself is a good example of how opportunistic routing can be effective, and striving for global optimisation for every route would not be practical, as the problem is NP complete [4, p130]. The deficiency would be better overcome by designing for efficiency and allowing independent routing and route optimisation services to set up or compute routes offline, and feed run-time input to the translation process. This is just what is being done in the Internet by backbone and access service providers, so that the virtualisation remains consistent with current practice and experience. However, the activity is now partitioned from networking, and without e2e addressing, there might be room for more providers and better overall service.

2.1.3 Security by invisibility

The absence of e2e addressing presents new opportunity for security, because it would be possible to provide e2e connectivity to client applications without revealing the server hosts. For this to work, an indirection must be built into the virtual-to-real translation away from the client's network neighbourhood.

It should be possible to adapt secure transaction protocols to virtual paths, because their security depends on the e2e connection and not on the available of e2e address.

These issues are discussed in more detail in Section 3.5 with reference to the present architecture.

2.1.4 Unification with telephony

Two features qualify telephone numbers as virtual addresses: they are arranged geographically or by service category (e.g. the US "800" service), rather than by network location, and the service is provided by virtual circuits. The idea of server and client requests physically meeting within the virtual space, is also exactly followed, because by subscribing, a user in effect advertises his or her accessibility in the telephone system directory. Unification with telephony thus seems more natural with address virtualisation.

This does not mean that Voice-over-IP (VoIP) and related protocols would be obsoleted, because these are required for a different reason, viz that it is difficult to support multiple media and data streams robustly and efficiently in the cellular devices. Rather, as pointed out in Section 1.1, the address virtualisation would allow the IP based solutions to be deployed more rapidly, and in a more appropriate, lasting manner, as explained in Section 2.1.6 below.

2.1.5 Domain transparency

As explained in Section 1.2, the virtualisation necessarily changes the API, because networking APIs like the BSD sockets were based on the premise of real network addresses. Though virtual addresses can be transparently accommodated as yet another address family, the usage and management of application data and services are clearly affected.

As depicted in the architecture overview, Fig. 1, the impact would be both one-time and for the better, allowing "enabled" applications to continue to operate over the existing LANs and the Internet, as well as connect to other domains that do not share the Internet address space, including native ATM networks and IP islands.

There is admittedly little difference from existing NAT and virtual private network (VPN) technologies at the network level with regard to hosting IP islands, as the virtual path mechanism involves a similar translation of address labels. The gain lies in supporting very large scale services, which cannot be done with NAT or VPN, as explained in Section 1.4.

Additionally, there is substantial gain to be had in the application space from the simplification of data location and accessibility management, which would no longer depend on the nuances of individual network configurations.

Incidentally, the virtualisation also provides opportunity to simplify and generalise the API to transparently handle native interfaces to diverse media such as raw Ethernet or MPLS, and to support distributed parallel applications efficiently and with greater elegance, with appropriate device driver software in each case (Section 2.5.1).

2.1.6 Mobility

The telephony equivalence means that the SS7 signalling techniques would be readily applicable to any virtual addressing framework for ensuring mobility. However, a principal reason for migration to IP in

cellular services is the difficulty of managing multiple streams, of audio and video along with data, in the handheld devices, along with access to the existing Internet.

The problem is also being addressed by the emerging MPEG-7 standard, among other solutions, but there are likely to be cases where IP routing would remain preferable. This is currently possible using NAT or VPN, but only for small cellular service operators, as pointed out in Section 2.1.5. Migration to IPv6 would solve the address problem, but cannot assure compability with the large body of applications currently written for IPv4.

Address virtualisation, on the the hand, would be permanent and indefinitely scalable solution, with a more user-friendly network interface at the same time.

2.2 Principles of design

The fundamental problem to be addressed by the present architecture, as explained in Section 1.1, is how to provide e2e connectivity without requiring the application clients to identify, and optionally without allowing them to identify, their server hosts. This requires some form of a virtual address space, and automatic translation to virtual paths within the network.

2.2.1 Reversal of roles

Given the translation and the e2e principle motivating elevation towards the host application level, it becomes natural to identify the virtual addresses with services provided by application servers. More particularly, e2e addressing has hitherto involved two levels of indexing, over the network to the host and on the host, to a service port, i.e. in the form <host-location, service-port>. The service must come first for virtualisation, essentially reversing the indexing order to the form <service-id, host-component>, where the latter is of significance only at the application server end and in distributed parallel applications, as described in Section 2.5.1.

2.2.2 Service indirection

As pointed out in Section 2.1 and Section 2.1.3, the best that can be done is an indirection in the virtual space accessible to the host applications, and the indirection must be built into the translation,

requiring the latter to be distributed. The indirection is readily achieved by having the application servers first advertise in the virtual service-id space, and having the client connection requests meet half-way, as it were, at the points of advertisement. The e2e connectivity is then conceptually obtained by simple concatenation of the physical paths taken by the advertisement and connection requests.

2.2.3 Route-interpretation of user request

Additionally, both the e2e connectivity and the translation needs to be as efficient as IP routing, as explained in Section 2.1.2, whose efficiency comes from address classification and aggregation, allowing higher order address bits to be mapped to the next hop. In effect, IP owes its efficiency to "route interpretation" of the address bits, and this strategy is ideally adopted for the physical path traversals by arranging the virtual space hierarchically, which also provides the convenience of URLs and presents a spanning tree constructed by the applications themselves, which is opportunistically exploited in the route interpretation.

2.2.4 Homogeneised URLs

The URLs differ from those used on the World Wide Web in that the principle of addresslessness prohibits distinction between host locations and services. They are therefore homogeneised and resemble pure file system paths, so that any trailing portion of a URL may turn out to be hosted at a single host, using a modified web server, within the present framework. The fundamental distinction from current practice is that the URLs are now used for all connections and not limited to TCP/HTTP and related protocols.

To summarise, the present architecture is based on the following key ideas:

- virtual addressing in the form of a distributed, hierarchical services name space forming a control plane above the network,
- e2e logical connections over the name space, exploiting its spanning tree property, by concatenating the physical routes taken by server advertisement and client connection requests,
- efficient routing for the logical connections by interpretation of URLs to the advertisement points,
- distributed translation from the e2e connections to virtual paths within the network,

- and importantly, locality of addresses and network information in both control and network planes.

Additionally, as observed in Section 2.1.2, the locality property allows independent offline route set up and optimisation services.

2.3 Addressless connectivity

```
        /-----\                            /-----\
        |  B  | <------------------------> |  F  | <---+
        \-----/                            \-----/     |
           ^   \----\             /-----/   ^          |
           |         \           /          |     /-----\
           |         <<q>>====<<r>>         |     |  G  |<--+
           |           ||      ||           |     \-----/   |
           |           ||      ||           |               |
       /-----\         ||      ||        /-----\         /-----\
       |  A  |         ||      ||        |  E  |         |I    |
       \-----/         ||      ||        \-----/         |  (x)|
         ^    \------<<p>>     ||  /-----/  ^            \-----/
         |             ||     <<s>>         |
         |             ||      ||           |
         |             ||      ||         --------
     -------           ||      ||         |     b|
     |a    |<===========//     \===========>|---+  |
     | +---|                              | v |  |
     | | u |                              --------
     -------
                                       /--\     addressless
  ||,<====>: data links                |  | : connections server
     ----->: control configuration     \--/
     ----- : signalling links          ----
       (.) : service access point      |  | : application host
     <<.>> : switch                    ----
```

Fig. 5: Control and transport planes

Accordingly, Fig. 5 depicts a sample implementation, comprising a control plane of Addressless Connection Servers (ACS) geographically distributed and configured in a hierarchy; the network plane composed of switches, data links and the host interfaces; and signalling connections between the ACS nodes and the switches. The ACS nodes are labelled A through I, the switches, p through s, and the hosts, a through c. The URLs will be given in the familiar format //<name>/<name>/.../<service>, where the leading "//" designates the root of the hierarchy, and each <name> denotes an ACS, except the

last, which represents an advertised service. This notation scheme is adopted here purely for the purpose of describing the architecture.

2.3.1 End to end connectivity

The illustrative example concerns an application server process u on host a advertising its service at the homogeneised URL //F/G/I/x, and an application client process v executing on host b requesting connection to this service, citing the same URL //F/G/I/x to its operating on host b. Given the hierarchical relation as shown, it is straightforward to propagate either request in the control plane, and to simultaneously construct a reverse URL to the requesting host and process, until the request message reaches the ACS I. Here, u's request results in creating a symbolic name x and binding the reverse URL //B/A/a/u, while v's request results in look up of the name x and retrieval of the bound reverse URL, //B/A/a/u, so that the client (v's) request can now be further propagated all the way to the server process u on host a. It is a trivial matter to cache "downhill lookups" so that successive client requests reaching the ACS F can be sent directly over to ACS B, avoiding a two-way traversal to ACS I and the overhead this would entail at ACS F. Thus, using only a control plane of name servers and hierarchical URL interpretation, e2e connectivity has been obtained in this example without requiring e2e network addresses.

2.3.2 Comparison with DNS/IP

The URLs are addresses of services, not of the providing hosts, and are not real network addresses in this sense.

However, as remarked in Section 2.1, the URLs are not totally virtual, as they do locate the ACS hosting the URL, I in the example. If the reverse URL bound at I, //B/A/a/u, were to be returned to host b, the e2e connection could be not only established directly from b with almost the same efficiency, but it could be cached at host b or at the next ACS, E, the same way as DNS lookup results are currently cached.

The reverse URL //B/A/a/u is a compact version of a hop-by-hop route to the server host a, making it a real address. There is still some indirection in the sense that the URL is not a simple name identifying the final destination, but it would be clearly preferable not to make the reverse URLs accessible to clients in interest of server security.

The control plane is clearly sufficient for paging or short message services (SMS), as the message could be carried as payload with the

client request packet. Optionally a reply can be returned as payload with the server's acknowledgement, following the reverse URL to the client, constructed during the forward propagation.

This is favourable as it eliminates a separate DNS lookup. The saving over IP would not be much in web usage from conventional computing hosts, for instance, where the host DNS cache misses are typically small [6] but it would be useful for small devices with little cache of their own.

2.3.3 Relation to Web servers

As mentioned in Section 2.2, any trailing segment of a URL may be transparently hosted on a single physical host, trading computational load for network delays. For example, the ACS nodes G and I could be merged, so that the URL segment /I/x could be mapped to a file x in a directory I on ACS G.

Existing web server code may thus be combined with existing DNS code for implementing the ACS.

2.3.4 Similarity to telephony and email

Note that both notions of application processes and file table entries in Section 2.4.1 are conceptual, as the signalling process could be identically employed even without conventional computing hosts.

For instance, in voice telephony, both hosts a and b could be telephones, and the calling process v would then be the human user, the waiting process u would be the ring circuit on the b, and would have conceptually advertised its telephonic URL as //1/914/945/2934 say, in the global telephone directory referenced by the existing SS7 signalling procedures.

Email addresses are also functionally equivalent to the service access points of the present framework, and the three are more closely associable with the care-of addresses in Mobile-IP than with static network locations ordinarily signified by IP addresses.

2.3.5 Locality in control plane

The ACS links would be statically set up and only the hierarchical relations, which are again local (parent-child), are used for the e2e connection, hence there is no dependence on e2e addressing.

The above example is thus fundamentally significant because globally unique addressing had been hitherto considered axiomatic.

2.3.6 Open extensibility

As in the DNS, the hierarchy allows the control plane to be readily extended by adding subtrees, the only constraint being that each subtree must have a unique name under its parent.

2.3.7 Efficiency of routing

In ATM source routing, which reflects the premise of e2e addressing but not "route interpretation" (see Section 2.2), a search space of size (b**n) is involved, where b is the mean branching factor (out-degree) and n is the number of hops to the destination. This characterises the size of the search database needed at each node for route computation, and the PNNI specification calls for hierarchical organisation to keep this manageable. To compare, hop-by-hop routing incurs a next-hop address space of size b per hop, for a total space size of only b * n, although it has the possibility of looping. The same holds for ideal "route interpretation", as in IP before CIDR.

The control plane routing is equally efficient because each step along the hierarchy corresponds to a branching of the ACS tree, and the efficiency is owed at least partly to the fact that the latter is a spanning tree. However, as remarked in Section 1.1, dependence on a single spanning tree has its shortcomings, as follows:

- The control route depends on where the service is advertised with respect to the prospective client host.

- The control plane links may not cover the shortest network paths, making optimality unlikely.

- Even when close to the shortest network path for a given client, the advertisement may lie on a different branch, as illustrated in the above example by binding on ACS I instead of F, E, B or A, any of which would have kept the e2e path optimal at least within the control plane.

- The hierarchy forces top level ACS nodes to be involved more often. Without caching or offloading, the root node capacity would limit the scalability.

The open extensibility and the choice of advertisement location guarantee that the first two defects can be overcome in any specific case. Also, there is additional freedom in the translation to virtual paths, which need to be optimal, whereas connection latency is more important for the control plane routes (Section 2.1). The remaining defects are addressed in the following subsections.

2.3.8 Stateless interpretation

The basic algorithm of Section 2.3.1 is of itself stateless, as no reference needs to be held at any ACS, so the URL interpretation is the main delay.

It will be shown that statelessness can be maintained even with optimisations (following subsections) and in the translation to virtual paths (Section 2.4).

2.3.9 Control plane optimisations

The control plane is meant for the initial e2e connectivity, somewhat like the DNS, or for one-time short messages for which a virtual path is not required. This makes efficiency the main concern, to avoid holding the request state at any ACS for too long (Section 2.1).

Off-branch lookups, like ACS I with respect to host b in the foregoing example, may be cached at the branch point ACS (F), avoiding two of three passes through it (to I and back).

The hierarchy itself is conceptual - there is opportunity for building translation rules at lower level ACS nodes to exploit additional control path links, or "shortcuts", through the ACS tree. In the above example, a shortcut link from E to A would be well poised to avoid going through the roots F and B altogether, provided E can predict that the reverse URL to be fetched from I will take the request through A.

This too can be arranged if the reverse URL can be percolated back towards the client host for some distance, although leaking it to the client host everytime would burden the control plane unnecessarily and lose security advantage (Section 2.3.12). A controlled back-propagation is possible and is described next.

2.3.10 ACS shortcuts and forward-encoding

The effective traversal path in the above example is v-b-E-F-B-A-a-u, and the strict hierarchical interpretation would miss a shortcut link

E-A, assuming it to be available. If the route were being computed at a single node, it would be a simple matter to scan the traversal path string for the pattern "E-*-A" and replace it with "E-A". This may be represented by the production

$$E * A \mapsto EA; \qquad (1)$$

similar production rules will be applied to describe the translation to virtual paths. In general, only regular expressions are involved, so the grammar representation is more for convenience than necessity.

Since the interpretation is distributed and each hop costs network delay, which can never be eliminated, backtracking is to be avoided or at least kept out of the critical latency path. One way to achieve this is by "forward-encoding" as follows:

> When sending the request from E to F, tag the production (1) with the request. The pattern would be matched at F, where the reverse URL to the server host becomes available. F may then back-propagate the request together with reverse URL to E for routing through the shortcut.
> E may cache the result to avoid referencing F in future requests to the original URL //F/G/I/x.
> F may decide to continue with forward-propagation if the cost of back-propagation would be greater. However, caching at E can be more effective in cost-reduction in many applications, especially when host b has low mobility. Both options can be exercised simultaneously by allowing F to propagate the request both ways, while annotating the back-propagated message with a flag to alert E that the message is for caching only.

Several variations are possible in this approach. For example, instead of encoding the production (1) literally, it would be sufficient to encode just the depth of ACS E in the tree along with the identity of ACS A. Even this could be omitted and F requested to back-progate only anyway; ACS E then gets to make the best choice for the onward route. Yet another variation is to encode only the depth, and leave it to ACS F to determine if whether the backtracking depth would be excessive.

Additionally, it might be noticed that the ACS tree is effectively a cache-hierarchy in reverse, and other cache management techniques from the DNS experience, as well as from CPU architecture and distributed shared memory (DSM) fields, may be applied in the present context.

2.3.11 Intelligence and admission control

Besides interpreting and caching URLs, it would be obvious that server-configurable functionality could be added the advertised name bindings like x in the above example, to be automatically invoked using the client's reverse URL as argument. Three typical uses would be:

- Service zoning and redirection - by returning a different server-locating URL depending on the client's location in the ACS tree.
  This could be used, for instance, for customising the service by language or geography, or for individual corporate customers, e.g. like medical insurance organisations (HMOs in the US).

- Automatic signalling for setting up peer-to-peer paths in symmetric parallel applications.

- Client authentication - unauthorised clients can be denied access to the server host by the ACS node refusing to return or forward the referenced server URL. Authentication keys can be stored with the bindings for this purpose.
  This differs from current IP security, where the server host is always vulnerable. For example, SYN attacks are readily mounted on Web servers because the server hosts are visible independently of the logical (TCP) connection.
  See also Section 2.3.12.

- Back-propagation to trusted hosts - which would be useful for framework administration.

2.3.12 Security and resilience

It might be thought that since the ACS tree is evidently the sole means of connectivity, it would be more vulnerable to attack than IP.

Section 2.1.3 and 2.3.11 imply that the vulnerability would be limited to the local addressing islands hosting the ACS nodes and to the physical links between them.

Instead, the fact that the ACS is the only means of access also protects it against attacks from within the framework. Consider, for example, that a denial-of-service attack on ACS node F by a truant host b would have to pass through its provider ACS E, which would likely get choked first, cutting off the attacker. This works equally well if the attack is mounted from multiple hosts.

Denial of service attacks against application servers have become routine in the Internet, but they are readily prevented in the present framework by simply keeping the server hosts invisible.

2.3.13 Prediction database

An alternative method, which can be applied in conjunction with the above, is to build up the prediction database at lower ACS nodes like E anyway, ahead of time. This can be done by propagating the binding information from ACS I periodically to the neighbouring ACS.

2.3.14 Optimisation by k-locality

As illustrated by the regular expression reduction rule (1), the above techniques generally allow "peephole optimisation" of the basic traversal path v-b-E-F-B-A-a-u. The peephole optimisation however makes the interpretation at each ACS node computationally more expensive, by increasing the search space bound from b to (b**k), where k is the size of the peephole window, as shown in Fig. 6 below. Multiple and overlapping shortcuts, for example as shown in Fig. 7, would also be readily handled by peephole optimisation.

```
                |<--- k --->|
                /-----------\
                |           |
        v - b - E - F - B - A - a - u
```

Fig. 6: Peephole optimisation

```
                /-------\
                |       |
        v - b - E - F - B - A - a - u
                    |       |
                    \-------/
```

Fig. 7: Overlapping shortcuts

Advantage lies, of course, in the reduced number of hops, yielding a total search space size of [n/k](b**k), which reduces to n*b comparable to IP routing when k = 1, and increases to (b**k), equivalent to that in the source-routing problem. The present approach thus yields a continuous range of tradeoffs in route computation efficiency between hop-by-hop and source routing.

It should be noticed that even at k = 1, the approach is superior to hop-by-hop routing in both IP and ATM, in that looping is inherently avoided because the ACS is already a spanning tree.

Clearly, the process cannot guarantee global optimality, but latter is an NP complete problem ([4, p130]) and cannot be solved in practice in any case. Instead, like greedy algorithms, the present architecture offers, as previously remarked (Section 2.1, Section 2.3.6, Section 2.3.7), efficiency coupled with opportunistic freedom that might more than make up for not attempting global optimisation at all.

2.4 Addressless transport

The logical end-to-end route obtained above may be thought of as an end-to-end cable in suspension bridge construction, as it allows the data paths to be set up in the transport plane by broadside-on signalling to the switches, preserving the O(n) efficiency, given a rough geographic correspondence between the switches and the ACS as suggested in Fig. 5. In the example of Section 2.3.1, the client's request traverses the control path v-b-E-F-B-A-a-u, after deletion of the common retraced part F-G-I-G-F, which could be avoided in any case by caching at ACS F. For the distribution of switches given in the example, a basic one-pass signalling sequence to construct the virtual path u-a-p-q-r-s-b-v is given in Figs. 8 and 9 below.

2.4.1 Two-way path stitching

The sequence constructs a two-way virtual path simultaneously with the connection request. This is not the best way (Section 2.4.2), but serves to illustrate the information flow.

First, when host b forwards the request from process v to ACS E, it would have already set up a file table entry both as a placeholder for the response and for the return virtual path on success. This entry has an index say 5. Host b passes this index along with the request URL //F/G/I/x and the identity of the requesting process v. The caller identity is symbolic and included solely for illustrating the reverse URL construction: the latter is used for paging replies, as already stated, but only the index (=5) is actually needed for delivery to process v.

ACS E determines from the forward URL //F/G/I/x that the request needs to be percolated upwards to ACS F. It also determines, in this basic signalling scheme, that the data paths must be routed via switch s, using its network knowledge as described in Section 2.4.4.

```
                                                      //F/G/I/x,b.5,v
                  A           B            F           E <---------+
                                                                   |
      a           p           q            r           s           b
   -------     -------     -------      -------     -------     -------
1  |     |     |     |     |     |      |     |     |     |1    |v:   |5
   | u   |3    |     |     |     |      |     |     |     |     |     |
   -------     -------     -------      -------     -------     -------

                                          //F/G/I/x,s.1,b/v
                  A           B            F <-------- E
                                                       |
      a           p           q            r           s           b
   -------     -------     -------      -------     -------     -------
2  |     |     |     |     |     |      |     |2    | b.5 |1    |v:   |5
   | u   |3    |     |     |     |      |     |     |     |     |     |
   -------     -------     -------      -------     -------     -------

                            //B/A/a/u,r.2,E/b/v
                  A           B <--------- F           E
                                           |
      a           p           q            r           s           b
   -------     -------     -------      -------     -------     -------
3  |     |     |     |     |     |      | s.1 |2    | b.5 |1    |v:   |5
   | u   |3    |     |     |     |      |     |     |     |     |     |
   -------     -------     -------      -------     -------     -------

                A/a/u,q.8,//F/E/b/v
                  A <-------- B            F           E
                              |
      a           p           q            r           s           b
   -------     -------     -------      -------     -------     -------
4  |     |     |     |     | r.2 |8     | s.1 |2    | b.5 |1    |v:   |5
   | u   |3    |     |     |     |      |     |     |     |     |     |
   -------     -------     -------      -------     -------     -------

      u,p.7,//F/E/b/v
      +<--------- A           B            F           E
      |           |
      a           p           q            r           s           b
   -------     -------     -------      -------     -------     -------
5  |     |     | q.8 |7    | r.2 |8     | s.1 |2    | b.5 |1    |v:   |5
   | u   |3    |     |     |     |      |     |     |     |     |     |
   -------     -------     -------      -------     -------     -------
```

Fig. 8: Stitching algorithm: request pass

```
     //F/E/b/v,a.4
      +---------> A           B            F           E
      |           |
      a           p           q            r           s           b
   -------     -------     -------      -------     -------     -------
6  |u:p.7|4    | q.8 |7    | r.2 |8      | s.1 |2    | b.5 |1    |v:   |5
   | u   |3    |     |     |     |      |     |     |     |     |     |
   -------     -------     -------      -------     -------     -------

                 //F/E/b/v,p.8
                  A --------> B            F           E
                  |
      a           p           q            r           s           b
   -------     -------     -------      -------     -------     -------
7  |u:p.7|4    | q.8 |7    | r.2 |8      | s.1 |2    | b.5 |1    |v:   |5
   | u   |3    | a.4 |8    |     |      |     |     |     |     |     |
   -------     -------     -------      -------     -------     -------

                                E/b/v,q.9
                  A           B ---------> F           E
                              |
      a           p           q            r           s           b
   -------     -------     -------      -------     -------     -------
8  |u:p.7|4    | q.8 |7    | r.2 |8      | s.1 |2    | b.5 |1    |v:   |5
   | u   |3    | a.4 |8    | p.8 |9      |     |     |     |     |     |
   -------     -------     -------      -------     -------     -------

                                             b/v,r.3
                  A           B            F --------> E
                                           |
      a           p           q            r           s           b
   -------     -------     -------      -------     -------     -------
9  |u:p.7|4    | q.8 |7    | r.2 |8      | s.1 |2    | b.5 |1    |v:   |5
   | u   |3    | a.4 |8    | p.8 |9      | q.9 |3    |     |     |     |
   -------     -------     -------      -------     -------     -------

                                                           v,s.2
                  A           B            F           E --------->+
                                                       |           |
      a           p           q            r           s           b
   -------     -------     -------      -------     -------     -------
10 |u:p.7|4    | q.8 |7    | r.2 |8      | s.1 |2    | b.5 |1    |v:s.2|5
   | u   |3    | a.4 |8    | p.8 |9      | q.9 |3    | r.3 |2    |     |
   -------     -------     -------      -------     -------     -------

               Fig. 9: Stitching algorithm: acknowledgement pass
```

ACS E accordingly first signals to switch s, telling it the return destination (host b) and index (5) for setting up the return path table entry as shown in the figures. Switch s returns the index, say 1, of the new entry, to E, which then passes it along with the request URL and the growing reverse URL b/v. ACS F determines the onward path either from its cache or by querying ACS G, obtaining the reverse URL //B/A/a/u, where the last symbol u similarly refers to a file table entry on host a where the process u is presumably waiting to accept the connection.

The sequence continues as shown until host a is reached and process u accepts, resulting in a new "accepted" file table entry for the new forward path at index 4, which is now returned together with the OK response to ACS A, which in turn passes it to switch p along with the source identity a. These values are used to fill a new forward path entry at index 8 on switch p, and the index is once again returned to the signalling ACS, A. The sequence continues until the OK response reaches host b, where the next switch identity s and index (=2) are entered into the file table entry initially allocated to complete the forward path, before handing the response to v.

2.4.2 Delayed signalling and asymmetry

The preceding scheme is not efficient because ACS-switch signalling delays are incurred even on the forward pass, and the return path entries in the switches would have to be torn down in the event of failure, say if process u had gone away when v's request arrived, which would not only slow the response but would place additional signalling load on both the switches and the ACS.

The solution is to omit the return path signalling, constructing only the forward path with the returning acknowledgement, because the return path can be subsequently set up, if needed, by inband signalling in the forward path, further reducing the signalling load on the control plane.

The return path set up request can be deferred and combined with the first data packet from the client if the application protocol requires the client to send the first message, as in HTTP, or initiated immediately on receiving the acknowledgement if the server is expected to transmit first, say with a login prompt. In many cases, for example in bulletin services "pushing" news clips to registered clients, the return path might be avoided altogether.

2.4.3 Stateless signalling

The two-way algorithm of Section 2.4.1 does not need to impose signalling state in the control plane as the acknowledgement messages are routed by the reverse URL constructed in the forward pass and the return path switch entries for teardown signalling, to be used in the event of failure, could be carried along with the reverse URL.

With deferred return path signalling (Section 2.4.2), the return path entries information would also be avoided.

2.4.4 Inclusion of network state

As mentioned in Section 2.3.10, the topology information is readily represented by production rules of the form

```
                              bEF  |-> bsF
                              sFB  |-> srB
                                                          (2)
                              rBA  |-> rqA
                              qAa  |-> qpa
```

which describe the translations successively applied at ACS E through A, in effect replacing themselves, as nonterminals, with the switch names s...p, respectively, which, together with the end hosts a and b, constitute the terminals for the virtual path. Although shown bundled together, only the relevant rules need to be available at each ACS.

In the example of Section 2.4.1, the successive transformations are

```
                     vbEFBAau   |-> (at E) vbsFBAau
                                |-> (at F) vbsrBAau
                                                          (3)
                                |-> (at B) vbsrqAau
                                |-> (at A) vbsrqpau
```

As noted in Section 2.3.14, the translation grammar is restricted to regular expressions, because the nonterminals (ACS) cannot even be recursively applied, as recursion can only signify loops. In effect, this formally proves the inherent avoidence of loops claimed in the present architecture (Section 2.3.14).

Each of the above rules involves both pre- and post-context, which is needed because the choice of switch must depend on both contexts. This makes the search space size b**2 at each ACS, where b denotes both the in- and out-degree mean values, and changes the overall search space size to [n/k] b**(2k) (see Section 2.3.14).

The rule contexts are also conceptually useful as placeholders for the peer linkage data. For example, the post-context (A-a) in the fourth rule above conceptually represents the linkage data p:7 and a:4 in steps 5 and 6, of the basic stitching algorithm shown in Figs. 8 and 9, respectively.

2.4.5 QoS attributes and alternative paths

A straightforward extension is to append latency, capacity, bandwidth, current loading, cost and other possible attributes to each of the productions, generalising the representation to an attribute grammar. For example, consider the rules

```
                   bEF  |-> bsF   [10 MB/s, 90% full]
                                                          (4)
                 bE * A |-> bspA  [100 MB/s, $1:10/min]
```

both in ACS E's topology database, meaning that the first production, which would presumably route through switch r (see Fig. 5), may be applied if the current request is for a low data rate connection, and the second production, using the direct, expensive link, is to be applied if the bandwidth requirement justifies the cost.

The decision however involves predicting the occurrence of ACS A in the control plane path, as considered in more detail in Section 2.4.6.

As is usual in grammar productions, it is possible to have more than one equally applicable production, for instance, say there is a parallel path between s and q going through another switch r', with very similar properties to the path through r, resulting in the following rules at ACS F:

```
                             sF B  |-> srB
                                                          (5)
                             sF B  |-> sr'B:
```

Which production gets applied is then up to ACS F. It is also possible to distribute the path both ways, as discussed in Section 2.4.7.

2.4.6 Forward encoding with translation

The second rule in the set (4), which involves "lookahead" to ACS A, can be readily applied if the onward translation to URL //B/A/a/u happens to be found in ACS E's cache. No further interpretive effort is required there is a shortcut control plane link to ACS B, as considered in Section 2.3.10, so that ACS F and B are not involved.

If the shortcut is unavailable, howver, there needs to be an indication to ACS F and B that interpretation is not necessary and the request packet should be simply passed onwards to ACS A. This is already contained in the grammar formalism as the request arriving at F contains the partially translated path b-s-p-A, in which the nonterminals F and B do not occur. Further translation is thus suspended till the request reaches ACS A.

What if there is a "cache-miss" at ACS E ? Several variations are possible in this case, as explained for shortcut control routing in Section 2.3.10, and the same choices would be obtained for the present translation by tagging the lookhead rule in (4) in place of the shortcut rule (1). In all cases where ACS E gets to make the final decision, the information at the time of decision is exactly as if there were a "cache-hit", and the preceding considerations are then identically applicable.

2.4.7 Parallel paths

If the network link between a pair of switches already supports parallel pathways to be load-shared in some way, it would be transparently supportable in the present scheme.

The scheme however allows parallel paths to be applied even when the network links do not themselves support parallel paths. For this, each of the linkages represented by the rule post-contexts (Section 2.4.4) needs to be expanded into a list, and each step in the stitching process would then involve multiple switches in parallel. This can be readily applied to the rule set (5).

Picking both of rules (4) is more tedious but the principle remains the same. In this case, the connectionrequest must go through the longest of the control paths and cannot use the shortcut control link.

Although it would allow a one-to-one correspondence between the connection request and the linkage data, thereby avoiding list representation, splitting and merging of the request itself is not preferable because it would require the merging ACS to hold the component request messages.

2.4.8 Locality of network information

Section 2.4.4 through Section 2.4.7 describe strictly "peephole" translation, but the rule contexts bridge between adjacent ACS along the control path.

For example, in the transformation sequence (3), when the switch

identity r inserted by ACS F must identify switch r as seen from ACS F, in order that F can meaningfully signal to the switch in steps 3 and 9 in the signalling process (Fig. 8 and 9). When passed over to the next ACS, B, the terminal symbol continues to represent the same switch, but as seen from B. In absence of e2e addressing, the two representations of the switch cannot be the same and a local address translation is involved, as indicated below by elaborating the second and third steps in (3):

```
                     vbs:FBAau   |->  vbs:F:rBAau
                                 |->  vbsr:BAau
                                                                    (6)
                                 |->  vbsr:B:qAau
                                 |->  vbsrq:Aau;
```

where s:F signifies that s is currently as seen from F, and s:F:r signifies that both switches s and r are as seen from F. In the second step, the request has been handed over to ACS B, with simultaneous translation of switch symbol r to r:B, and so on.

It is tedious but trivial to show these address translations in all of the preceding translation rules, but it should be clear that locality is not compromised by these address translations.

Shortcut transport paths introduce some degree of nonlocality. For example, the link between switches s and p in Section 2.4.5, rules (4) requires a switch reference translation going from E:sp to sp:A. Note that ACS F and B cannot be involved for this translation, the linkages must be set up with respect to the end points of each switch-to-switch link, and moreover, the request may be routed over a control plane shortcut directly to ACS A (Section 2.3.10).

More importantly, the shortcut cannot be represented in the rule bases of ACS A and E unless they are each directly addressible from the other in a form that allows comparison with the remaining rules at these ACS. Regardless of how the nonlocal addressing is implemented, it should be clear that the requirement does not exceed the k-locality allowed in the control plane (Section 2.3.14), as long as the shortcuts are finite.

Locality in the sense of finite neighbourhoods is thus preserved in the translation process. This allows everywhere-local management of routing rules representing the network topology and conditions by offline services, as envisaged in Section 2.1.1 and Section 2.1.2, further complementing the open extensibility of the control plane (Section 2.3.6).

The translation would be best built into the translation rules as part of the network information encoding. As this would be done offline, the translation would incur only a small "runtime overhead".

2.4.9 Passive and active aggregation

As explained in Section 1.3, the notion of hierarchical aggregation introduced in MPLS makes the scalability of virtual paths comparable to that of IP (Fig. 2), and is readily exploited in the present architecture for the same purpose.

The locality property allows the aggregation to be set up by offline services as well, making it transparent to the translation process. For example, the direct link between switchs s and p, introduced in Section 2.4.5 rules (4), could be merely an aggregate path over the same physical route s-r-q-p established by the offline services.

This is a passive form of aggregation support, because the framework itself does not need to react to network conditions. It should be noted, though, that the rule base at pairs of ACS, like A and E in the above example, could be modified by the offline services dynamically to introduce or delete such aggregate paths depending on the network conditions, so that passive support is not necessarily static.

Aggregation may be actively supported as well, as illustrated by the following augmented rule set for ACS E, suggesting that E set up and utilise an aggregate path depending on the traffic and cost:

```
              bEF |->  bsF   [10 MB/s, 90% full]
           bE * A |->  bspA  [aggregate, 100 MB/s, $1:10/min]        (7)
           bE * B |->  bsqB  [aggregate, 100 MB/s, $0:70/min]
```

It is assumed that the aggregate path would be automatically torn down by ACS E when the traffic falls below the same thresholds.

2.4.10 Transport security

Virtual paths as such provide a higher degree of security than possible in networks designed to be promiscuous like IP. While there can be no special privacy within a broadcast medium like twisted-pair Ethernet, over longer routes, virtual paths tend to be better because the packets will be delivered only according to the set up connections. This is difficult to ensure with IP routing, where packets can, in principle, show up at surprising places. Secondly, if a switch gets compromised, only the traffic through it would be exposed, but if an IP router is compromised, more damage could be inflicted by feeding false information to the routing protocols.

The corresponding vulnerability in the present framework is that of the routing rulebases. Since the rulebases are defined to be local in character and content, only traffic routed using the compromised rulebase should be affected. This would be considerably larger than the single switch vulnerability, but still a lot less than the possible damage from a compromised IP router.

In general, thus, the framework provides some security in addition to that currently obtained in IP by encryption.

2.5 Universal user interface

The user perspective is considerably simplified by the present framework, as all network connectivity is uniformly through URLs, and there is no representation of IP addresses or port numbers.

The application programming perspective is also simplified as all application connectivity is again through URLs. More importantly, new capabilities and issues arise as the e2e connectivity and routing functions are brought closer to the application layer, and the prototype system calls implementation on AIX4 is presented below to illustrate the differences.

2.5.1 Distributed parallel support

The simplification holds even for distributed parallel applications which currently use PVM, MPI or similar libraries, principally because the URLs are not translated, unlike DNS hostnames, to server host addresses. The URLs thus lose the point-to-point flavour of IP addresses, and can be treated instead as as abstract service entities equally suited for traditional client-server and for symmetric parallel applications.

Furthermore, as the virtualisation effectively inverts the indexing order from the form <host-location, service-port> to <service-id, host-component> (see Section 2.2), at both application servers and in parallel applications, the handle to the service in principle points to not one but a list of virtual paths, leading to the clients and peer processes, respectively. The handles were therefore referred to as contexts, reflecting this generalisation and to distinguish them from the BSD sockets, in the prototype code and documentation.

2.5.2 Interface route tables

Since each context handle potentially abstracts a list of virtual path terminations, their OS data structure may be simultaneously used in a computer clusters by switch fabric interface coprocessors as routing tables. More significantly, they can be aggregated into a form which is intuitive for user control (Section 2.5.3), so that a similar abstraction could be of use on cellular devices for handling multiple streams or conferencing connections.

2.5.3 IPC and DFS-like abstractions

An interesting implication is that a context handle can lead to peer handles on the same host, and can therefore be used as yet another inter-process communication (IPC) abstraction, like System V IPC and Unix sockets.

A related implication is that it becomes important to aggregate the path lists of contexts obtained from the same service URLs. The aggregate handles should have lifetimes extending beyond individual processes that create or use them; coupled with the IPC functionality, this suggests a System V IPC id-like representation on Unix hosts.

Additionally, most flavours of Unix now support a /proc filesystem providing a command-line interface to running processes, and an analogous /cxts filesystem of context handles appears appropriate, remapping their independent representation as IPC ids. This would further allow a control or query access to the individual virtual path terminations in a given context handle to be made visible as files in the /cxts directory.

In popular distributed file systems like AFS, NFS and DFS, the remote file trees are mounted onto the host file system under DNS names or IP addresses. The elimination of host names and addresses would make the file system more homogeneous, with little to distinguish the network part from the file system part, as remarked in Section 2.3.3. For example, "/.../a/b/c/d/x/y/z" could mean the file system pathname /d/x/y/z under the service location /.../a/b/c, or the pathname /y/z under /.../a/b/c/d/x.

2.5.4 Modified API arguments

While the uniform use of URLs simplifies the system calls, the generalisation makes it necessary to allow newer parameters and argument data structures to cater to the parallel applications. It was demonstrated in the prototype, however, that a single switch table interface was still possible to support the newer kind of "driver modules". In particular, sample client-server and symmetric parallel datagram interface modules were tested reusing the existing sockets implementation within the AIX kernel. See also Section 1.6.

2.5.5 Sample system calls

The AIX prototype implementation adds the following 5 system calls:

```
typedef int cd_t, fd_t;        /* context and file descriptors */
typedef char* URL_t;
cd_t context (int type, int mode, void* arg);
int cntl (cd_t, int cmd, void* arg);
int cbind (cd_t, URL_t, void* arg);
int cget (URL_t, int type, int mode, void* arg); /* cd_t or fd_t */
fd_t copen (cd_t, int oflags, void* arg);
```

The first system call context() creates a context handle, replete with IPC-like Unix permissions and mode properties. The second system call cntl() is functionally similar to the ioctl(), except that it operates on context instead of file handles. The third call, cbind (), would be invoked to advertise the service URL, as described in Section 2.3.1.

The fourth, cget(), returns a descriptor which is directly the file descriptor if the caller is a traditional client application, but is a context handle if either the caller is a server or the context belongs to a parallel computing application. In either of the latter cases, the approach avoids the need to make cget() calls, which would involve the ACS tree, multiple times to accept successive connections; instead, copen() may be invoked on the fetched context handle for this purpose, thus caching the previously fetched result in a visible way.

The extended POSIX system calls readx()/writex() are used for messaging in the multipoint (parallel) contexts, using the last argument as the peer index, following MPI philosophy. The file descriptors are as usual closed using the existing close() system call or automatically on process termination, and the cntl() call is used to destroy the context handles, because it is not always necessary that they should disappear automatically on last close.

Lastly, the type argument is used to specify the application paradigm, for example, whether client-server or parallel, and so on, in addition to the nature of the traffic (i.e. stream, datagram, etc.), generalising from the socket-type argument used in the BSD socket API.

3 Summary

3.1 Features and performance

The present framework is the first ever formal solution of the networking problem as an application-to-application connectivity issue (Section 1.2), and the generality is reflected in the unification possible in the user and application programming interfaces at the client hosts (Section 2.5). The approach fundamentally separates e2e connectivity from e2e addressing, showing the latter to be unnecessary at all levels, contrary to the basic assumption in all prior networks.

The framework further provides efficient if opportunistic routing, the efficiency principally accruing from interpretation of user or application supplied (virtual) address information in the form of service URLs (Sections 2.3.1,2.3.7) and translating, where necessary, from the interpreted control path to virtual paths within the network plane (Sections 1.5,2).

It has further been shown that the framework is inherently scalable, unlike IP and other networks like ATM, which have "hardwired" address space bounds. More particularly, virtual paths are advocated in the framework because they are by principle infinitely extensible, and the switching was shown to be logarithmically scalable by exploiting hierarchical aggregation of paths, as already defined in the MPLS architecture (Section 1.3). Correspondingly, while the ACS tree is of course hierarchical, and would allow application services to be advertised close to the direct paths to the clients (Section 1.5), shortcut links and DNS-like caching methods were shown to be possible that would reduce the load on the root nodes and make the framework as scalable as the DNS (Sections 2.3.9,2.3.10).

Since the framework uniformly depends only on local addressing (Section 2.3.14) and local network information (Section 2.4.8), it eliminates the need for network state "omniscience" (Section 1.1) required at each router in IP (Section 1.3) and near each switch in ATM PNNI for route computation, allowing unprecedented freedom for adding to the ACS tree as well as to the network plane.

3.2 Deployment and coexistence

The framework makes it easy for more providers to join in, and opens new opportunities for providing shortcuts links and routing rule databases (Sections 2.1.1,2.3.6,2.4.8).

A major incentive in the framework is that it adds to rather than replace the existing networks and the Internet, and the addition is above these networks (Fig. 1,Section 2.1.1), making it possible for application vendors to deploy their own implementations of the ACS tree and routing rulebases simultaneously over the Internet.

Further, it is trivial to map even the existing Internet at one or more points in a given vendor's ACS tree, e.g. mapping http://www.ibm.com to //us/ny/brewster/oakstview/com/ibm etc., providing full Internet access to a large number of "enabled devices" without allocating IP addresses (Section 2.1.6). The framework was also shown to be superior to VPN, NAT and related technologies in this regard (Section 2.1.5).

3.3 Open extensibility

The framework removes the e2e consistency requirement by ensuring that only local addresses are needed at all levels (Sections 2.3.5,2.4.8), making it easier to extend in both control and transport planes (Section 2.1.1). Naturally, compatibility is still required at the following interfaces: ACS-ACS, ACS-switch, switch-switch, ACS-host and switch-host, in addition to route computation and rulebase update (Section 2.4.8).

In principle, the provider of each ACS can dictate the registration protocol for attaching subtrees, as well as protocols for host registration, request propagation, caching and rulebase management.

3.4 Administration

Since e2e addresses are abhorred, the only sure way of administrating the framework is by local access to the ACS and the switches.

This is also the principal vulnerability, but securing the local access allows far better security than possible in IP (see below).

3.5 Security

The architecture offers better security than IP (Section 2.4.10) and can protect server hosts from denial-of-service attacks by making them invisible to the clients (Section 2.1.3), as long as the reverse URLs are not allowed to back-propagate all the way to the client hosts, which can be easily ensured by strict authentication of ACS nodes and

subtrees. The architecture allows additional defence by supporting in-network authentication before the client connections are granted.

The architecture is also designed to protect itself against denial attacks, because the nodes closest to the attacker would be choked off first (Section 2.3.12).

The security aspects appear to be superior, as is expected of a virtual addressing system (Section 2.1.3). However, they should be regarded as additional to, and not replacement for, the e2e encryption-based security already available or being researched in IP.

4 Intellectual property

The IETF is being made aware of an IBM patent application concerning the present architecture.

Acknowledgements

This proposal reflects insight and influence from many people over the past five years, notably (but not limited to) Nick Trio, Steve Deering, V S Pradeep, Ronald Mraz, Gabriel Silberman, Jonathan Shapiro, David Safford, Sonja Buchegger, Laurent Frelechoux, Sean Rooney, Annie Gravey, Udo Krieger, Marc Girouard and A Joseph Hoane.

Author's address

V Guruprasad ("prasad"),
IBM T J Watson Research Center,
Yorktown Heights, NY 10598,
USA.

Tel: +1 914 945 2934
Fax: +1 914 945 4077
Email: prasad@us.ibm.com, vgprasad@ieee.org.

References

[1] V Guruprasad. Who needs addresses? In IEEE European Conference on Universal Multiservice Networks, Colmar, Oct 2000.

[2] S King et al. The case for IPv6: draft-iab-case-for-ipv6-06.txt, 2000.

[3] W R Stevens. UNIX Network Programming. Prentice-Hall, 1990.

[4] C Huitema. Routing in the Internet. Prentice-Hall, 2nd edition, 2000.

[5] The Message Passing Interface (MPI) Standard. http://www-unix.mcs.anl.gov/mpi/, 1997.

[6] B Krishnamurthy and C E Willis. Analyzing factors that influence end-to-end Web performance. In 9th International WWW Conference, May 2000.