Assured Destination Binding: A Technique for Dynamic Address Binding

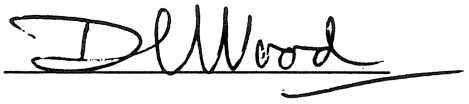
Robert H. Stine, Jr.
Paul F. Tsuchiya

March 1987

MTR-87W00050

SPONSOR:
Defense Communications Agency
CONTRACT NO.:
F19628-86-C-0001

The MITRE Corporation
Washington C^{3}I Division
7525 Colshire Drive
McLean, Virginia 22102-3481

MITRE Department
and Project Approval: 

ABSTRACT

Assured Destination Binding (ADB) is an approach to dynamic binding that could be used within a distributed system to disseminate and gather any information that can be bound to an identifying label. ADB has potential applications in many areas, including the distribution of public cryptographic keys and the locating of resources in a distributed system. The primary application expected for ADB, however, is address binding—the translation of names into addresses. Because ADB may be used to effect logical addressing, it also offers a solution for the mobile host problem and multiport network problems.

This report describes the components and functions of an ADB system used for address binding in a communications system. Because of the wide potential for applications of ADB, a general description of ADB's underlying concepts and major design alternatives is also presented.

Suggested Keywords: Address binding, Naming and addressing, Landmark routing, Logical addressing, Data communications.

TABLE OF CONTENTS

	Page
LIST OF FIGURES	vii
EXECUTIVE SUMMARY	ix
1.0 INTRODUCTION	1
1.1 Organization	1
1.2 Address Binding	2
1.3 Dynamic Address Binding	2
1.4 Name Servers and Dynamic Address Binding	3
2.0 ASSURED DESTINATION BINDING (ADB)	7
2.1 An ADB Problem Domain	7
2.2 ADB Concepts	8
2.2.1 Naive ADB	10
2.2.2 Limiting the Number of Servers	10
2.2.3 Controlling Server Allocation	12
2.2.4 Other Embellishments to ADB	12
2.2.5 ADB in Full	13
2.3 Specification of ADB Functions	14
2.3.1 VSID Assignment	14
2.3.2 Resolving Host VSIDs to Server VSIDs	16
2.3.3 Collecting Server VSIDs and Addresses	17
2.4 Integration of ADB Functions	21
2.4.1 Issues in Allocating Hosts to Servers	21
2.4.2 Approaches to Host Allocation	22
2.5 ADB in Changing Environments	23

TABLE OF CONTENTS (Concluded)

	Page
2.5.1 Required Adaptiveness	24
2.5.2 Adaptation Mechanisms	25
3.0 ISSUES FOR FURTHER STUDY	27
4.0 CONCLUSIONS	29
4.1 Summary	30
REFERENCES	33
GLOSSARY OF ACRONYMS	35

LIST OF FIGURES

Figure Number		*Page*
1	Interactions Between ADB Components	9
2	Assurance of Finding a Server	11
3	Resolution as Least Upper Bound of VSIDs	17
4	ADB with Generated Server VSIDs	18
5	Resolution Based on Server IDs	19

EXECUTIVE SUMMARY

INTRODUCTION

Assured Destination Binding (ADB) is a distributed, reliable approach to address binding. In simple terms, ADB is a technique for finding things, by translating the "names" of objects into the "names" of their locations. Unlike other address-binding schemes, ADB is robust enough to make this translation even if the locations of any objects (or, equivalently, the names of any locations) are subject to frequent, unannounced changes.

ADDRESS BINDING

ADB is an address-binding technique: it tracks the association between the names of things and their addresses. In a communications system, addresses identify network "locations" in the network's topology, while names identify the things at those locations. It is frequently desirable to reference objects by their names, but message delivery requires a destination's address. "Address binding" is the procedure that supports the translation of object names into addresses. More formally, "address binding" is a means of defining a mapping from objects' names to their addresses.

Because of changes in the communications system, the address of an object may change. If so, the binding between the object's name and address must also change. Such changes can be made in real time if address binding is deferred until shortly before message transmission. Binding techniques that can bind names to addresses during the message delivery process are referred to as "dynamic address binding."

ADB was initially designed to provide address binding for Landmark Routing, a distributed routing scheme for very large networks. The ADB approach, however, could be applied to many tasks that require finding objects in a distributed system. In data-communications applications, ADB could effect logical addressing, and thus offer a solution to the well-known "mobile host" and "multihoming" addressing problems.

There are existing systems known as "name servers" which, among other services, perform address binding. However, name servers have problems in performing dynamic address binding:

1. Querying name servers increases both overhead traffic and delay.

2. The name servers and the channels leading to them may become performance bottlenecks.

3. If there are few name servers compared to network objects, communications may be widely disrupted if a server crashes (a reliability problem).

4. If there are secondary or backup servers, there may be considerable overhead in transferring copies of a master database of bindings to them.

5. Existing name server systems do not adapt dynamically to objects undergoing address changes: updates to the attributes of objects, including network addresses, are entered manually by authorized individuals acting on behalf of naming authorities.

6. Current name servers are not adept at bootstrapping—the process of initial system configuration in which user agents find servers and servers find one another. Some rely on broadcasts, which limits them to environments where broadcast is supported (e.g., local area networks). Others assume that each user agent always has access to the address of at least one name server. Another assumption is that each name server at least has access to the addresses of its superior and subordinate name servers. This restricts them to use in environments where server addresses can be expected to remain constant.

7. Name servers are applications-layer entities; they are both inefficient and, because of protocol layering constraints, unsuitable, for translating logical addresses (which are identifiers from the interface between the network and transport layer) to network addresses (which are identifiers from the network layer).

8. Existing name server systems impose a hierarchical name syntax, in which the names of objects have a structure that reflects their assignment to a given naming authority. An object's name must change if it changes domain, which could cause disruptions in communications.

ADB is completely free from problems five through eight and is less affected than name servers by problems one through four.

ASSURED DESTINATION BINDING (ADB)

We will describe the use of ADB to support address binding in a communications system. In this example problem, users residing on hosts exchange messages by use of a communications system, which can, with some reliability, transfer messages from one network point of access to another. A user is merely a potential source or sink of messages; the host on which a user resides is visible to other users of the communications system. We do not propose here to solve the problem of differentiating users on a single host—we assume the communications system has completed its job when a message arrives at the destination host.

ADB uses identifiers whose bindings do not change. In our model, each host has an ID that differentiates it from all other hosts on the communications system. The host IDs are permanently bound to their hosts. In addition, host IDs are assumed to be well-known within the communications system; they serve as the handles that identify message destinations. This host ID is similar to a DDN X.25 logical address, since it is a

low-level identifier that distinguishes one host from another, but does not show a host's location.

Another facet of our model is that at any given moment each host has an address. Unlike an ID, a host's address may change over time. A message source will not necessarily have immediate access to the addresses of other hosts; finding these addresses is the purpose of our address-binding technique.

ADB follows the client/server model; that is, user agents query binding servers to obtain bindings for users. The binding servers are able to respond because each host in an ADB environment will have a "beacon" that transmits its current address binding to a server. A host will not usually be at the address of the server that caches its binding. User agents, in contrast, are at the address of the users they represent, and beacons are the address of the hosts whose locations they report.

The fundamental notion of ADB is that, based on a host's ID, beacons and user agents agree about which binding server is appropriate for caching that host's address binding. Beacons and user agents use the same process to transform a host's ID to the address of its binding server.

We will start with the simplest example of ADB. At each host—actually, at each addressable network object—we posit a binding server. Each host has a unique, enduring host ID, and an ephemeral current address. The address of the binding server that caches any one host's address binding is derived from that host's ID. The derivation of server addresses from host IDs has two steps. First, a host ID is transformed into a syntactically correct server address. The result of this transformation, however, will not necessarily be the current address of any existing binding server. "Assignment" is the name we give to the process of deriving a possibly "fictitious" server address. Next, the address that a host ID is assigned is mapped to the most similar current address of a real binding server. "Resolution" is what we call the translation to the address of a real server. Resolution assures that a host will have a binding server (hence the name, "Assured Destination Binding"). With ADB, an object's binding server can be located by use of the object's name. Hence, ADB provides a means to maintain connectivity across address changes in the communications system.

There are two points to emphasize in this transformation of a host's ID into the address of its binding server. First, the possibly fictitious address that the assignment function generates from the host ID is as durable as the host ID itself. Second, a host's binding server will change as current addresses change. If, however, beacons and user agents assign and resolve in the same way, then they should continue to agree about which current address is most like the address that the assignment function generates.

If there are fewer binding servers than current addresses, then resolution will be modified somewhat: the addresses generated by assignment would be allowed to resolve only to the current addresses of binding servers. Also, there must be some means of collecting and disseminating the current addresses of servers. There are a variety of approaches that could be used for this task.

There are potential problems in mapping host IDs directly into address space. This practice could lead to unfair loading of binding servers, which in turn could cause performance bottlenecks. This is because the current addresses of communications system are usually highly clustered, since addresses usually have structured formats. For instance, in the DARPA Internet, the first bit of a class-A address indicates address type, and the next seven bits indicate subnet affiliation. Hence, the first eight bits of all class-A addresses of ARPANET hosts are "00001010." Resolution of assigned addresses to current addresses would allocate fewer hosts to the servers whose addresses are within clusters than to outliers.

To ensure that order in the address space does not lead to unfair distribution of the binding server function, we propose that an intermediate level identifier be used. We call this identifier the Virtual Server ID, or "VSID." Functions similar to hash functions could be used to scatter both host IDs and the current addresses of binding servers into the VSID space. Resolution would then be performed to decide which image—which transformed value—from the current addresses of servers is closest to the image of a particular host ID.

Introducing VSIDs would help control how many hosts for which each server caches bindings. For networks not subject to address changes, however, mappings could be devised that send host IDs directly into address space in an even fashion. Or, host IDs could fill the role of VSIDs. Either approach might be incorporated into the design of simplified, efficient ADB systems.

ISSUES FOR FURTHER STUDY

There are topics in ADB that could be studied further. Certainly the functions and characteristics of ADB would probably be investigated at greater depth. Another area for future work would be to specify ADB at the level of detail required for its use in a communications system. In addition, a more thorough investigation into expanding the applications of ADB could be explored.

Before implementation, the design for various ADB algorithms and internal functions such as "assignment," "resolution," and "collection" should be analyzed. These procedures should be computationally efficient, yet they also must allocate server resources in a nondetrimental fashion. Further guidelines for developing these functions for particular operating environments would be useful.

The issues of integrity and trust within an ADB system should also be studied. Of particular interest are methods of authentication of binding updates, and also methods that might be used to restrict participation in address binding to trusted entities.

In addition to deeper investigation of the ADB functions, there are several areas in which the actions of ADB might be expanded. For example, if hosts are highly mobile, it is possible that ADB performance could be improved by predicting host motion; for example, anticipatory binding updates might be forwarded to servers. Clearly, this would require complex recovery mechanisms (e.g., what if a bad guess is made about the

host's path and velocity?), but it could improve the ability to reduce delay and maintain connectivity.

Once questions on ADB concepts have been answered, the task of developing a detailed specification for the use of ADB within a communications system could commence. ADB might be specified as an integral component of a newly designed communications system; no doubt this approach offers the best performance and versatility. Yet this is not the only possibility for fruitful use of ADB in a communications system. For example, ADB could introduce logical addressing within a communications system, thus increasing the system's robustness.

The concepts of ADB could be applied to areas other than address binding. For example, it seems feasible to use ADB to translate host aliases into unique IDs; server IDs could be assigned to aliases, and the ID-to-alias bindings cached at servers denoted by these server IDs. ADB might also be used to support generic naming for some services, in which the ability to request the first available server would be an asset. To use ADB in this fashion, it could be beneficial to have the binding servers also assume the function of resource allocators. In an even more general use of ADB for resource allocation, ADB could control access to resources and remote procedures in a distributed computer system.

Another interesting extension of the ADB approach would be using it to provide a means of decentralized public key distribution. In essence, ADB defines a means by which queries for information can rendezvous with answers. It would seem straightforward for a "key server" to be designed much as this document describes a binding server.

CONCLUSIONS

ADB has several points in its favor:

1. Other than uniqueness, it places no constraints on host IDs.

2. It is inherently robust and adaptive.

3. By support of logical addressing, it offers a solution to the mobile host and multihoming problems.

4. It is free of many of the inherent disadvantages of name-server systems.

ADB neither prescribes nor precludes any structure to the IDs of hosts. Hence, implementors are free to design a name space for their own needs, or use existing value for host IDs. An added advantage to the flat ID space of our scheme is that the recursive searches typical of hierarchical name spaces could be avoided. In a hierarchical address-binding scheme such as the Domain-name system, a name server might be required to query other name servers, and so on, to satisfy a binding request. This need for multiple server accesses could result in lowering performance of the address binding. In ADB, however, user agents would always transmit binding requests directly to the appropriate server.

ADB is also inherently fault tolerant. Because several servers may cache a host's address binding, and because resolution reassigns hosts to healthy servers if their servers fail, server crashes will not always preclude the servicing of binding requests. ADB also supports other tactics, such as migration, for achieving robustness. Given enough warning, moving hosts to differing network access points can lessen the loss of service caused by sites crashing. A primary difficulty in moving components of a distributed system, however, is the correction of the bindings. ADB allows users to maintain connectivity across address changes caused by host migration.

A further advantage of ADB is that it could support the translation of logical addresses to current routing addresses. Clearly, this capability offers a solution to the problem of mobile hosts connecting to a terrestrial network. In a nutshell, the mobile host problem is nothing more than the possibility that a host's address might repeatedly change.

The use of logical addresses also provides a consistent mechanism for handling multihomed hosts (i.e., hosts with two or more network addresses on the same communications system). Some multihomed hosts use each address for only certain classes of traffic, while others place no such restrictions on the use of their interfaces. In existing networks, directing traffic to the correct interface of a multihomed host requires specific local knowledge about that host. Logical addressing could give a systematic means for distributing this information. If a host's interfaces are restricted, then they might be given distinct logical names; a single host would then appear to be multiple network objects. On the other hand, if the interfaces are not restricted, then a beacon process might send one or both addresses to all its host's binding servers. Connection endpoints could then be defined by logical addresses, which would allow end-to-end connections to continue even one of a host's network interfaces fail.

Because the address binding service of our system is not limited to a few sites, many of the disadvantages usually associated with name-server systems could be avoided. For instance, if many servers participated in an address-binding system, each with a small purview, then high-volume data transfers typical of name-server systems would not be required. Also, use of many limited servers would render ADB less vulnerable than the name servers to widespread loss of service as a result of a single component failure. Also, congestion at the points of service would be avoided.

1.0 INTRODUCTION

Assured Destination Binding (ADB) is a distributed, reliable approach to address binding. In simple terms, ADB is a technique for finding things, by translating the "names" of objects into the "names" of their locations (actually, identifiers other than names can be used). Unlike other address-binding schemes, ADB is robust enough to make this translation even if the locations of any objects (or, equivalently, the identifiers of any locations) are subject to frequent, unannounced changes.

The core ideas of ADB were conceived by Paul Tsuchiya to solve the problem of address binding within communications systems that use Landmark Routing (Tsuchiya, 1987), a distributed routing scheme for very large networks. This document presents a general description of ADB's major design alternatives and underlying concepts; a detailed description of the application of ADB to Landmark Routing will be included in a later report.

To speak to the specific needs of Landmark Routing, and also to assist in describing the components and functions of ADB, this document focuses on ADB's application in translating host identifiers to network addresses. The ADB approach, however, could be applied to many tasks that require finding objects in a distributed system. For example, in data communications, ADB could be used to effect logical addressing, which in turn offers a solution to both the "mobile host" and "multihoming" addressing problems (Sunshine, 1981). In addition, ADB might be used for public key distribution, and for locating and allocating distributed resources and remote procedures.

1.1 Organization

The remainder of this section describes dynamic address binding, and evaluates the use of name servers in this role. The next section introduces ADB's major design features, along with their motivations. Afterwards comes a specification of the major ADB functions, with some thoughts on how these functions might be performed. Section 2 concludes with a discussion on the ability of ADB to adapt to changes in network configuration.

Section 3 outlines areas for future study of ADB, including potential uses outside the realm of address binding. Conclusions and a summary of this report are in Section 4.

1.2 Address Binding

ADB is an address-binding technique; it tracks the association between the names of things and their addresses. It is often said that an address in a communications system identifies the location of a network object, while a name identifies the object itself (Shoch, 1978). Finding an object requires that an association between these identifiers be made. Pairing the names and addresses of network objects is precisely what is meant by "address binding"; in broader terms, its purpose is to allow a translation between identifiers from different contexts. Phrased in the language of mathematics, address binding defines a mapping from the names of objects to their addresses.

Name-to-address translation can enhance access to the services of a communications system. For example, on the DARPA Internet, email identifiers (mailbox names) include a host-name component. Host names are much easier for people to use and remember than network addresses. To route messages, however, the DARPA Internet requires addresses. Hence, in the process of sending a message to a host, the name of the destination host must be translated into an Internet address.

For communications systems using logical addressing, address binding is a requirement, rather than an enhancement. So-called "logical addresses" are actually names, since they are independent of the location of network objects. Hence, there must be a means to map logical addresses to "network physical" (i.e., location identifying) addresses.

In our discussion, a "binding" is an ordered pair of identifiers from the mapping that the address-binding procedure defines. A binding is out of date or incorrect if its paired identifiers should no longer be associated (e.g., if a host changes its address, its host name should no longer be associated with its former address).

1.3 Dynamic Address Binding

Address binding can be done at system generation, at system initialization, or shortly before message transmission. The later or more "dynamic" the binding, the more gracefully a system can respond to address changes (Oppen, 1983). Dynamic address binding insulates users from network reconfiguration or the removal or change in location of network objects, and so makes a subnet more transparent to its users.

The primary disadvantage to dynamic binding is that obtaining a binding causes delay. This delay is particularly severe if address bindings must be transmitted over the communications subnet. The delays can be reduced, however, if often-used bindings are cached.

In many communications systems, address binding is only implicit, because so-called addresses also serve as host identifiers. For example, the 32-bit DARPA Internet "address" functions both as an address and a host identifier. The address function is clearly seen: an Internet address encodes the location of a host within the Internet topology. In addition, however, an Internet address is also employed as an identifier, distinguishing a destination host from others (Su, 1984). Probably the most frequent use of Internet addresses as host identifiers is in TCP connections, in which Internet addresses are used to determine the hosts where connection endpoints reside.

One problem with using a single value for an object's ID and address is that the entities of a distributed system usually will change locations more frequently than they need to change IDs (Oppen, 1983). Changes in the identifiers of visible network objects are disruptive to a communications system because they require changes to software and procedures. Meanwhile, it may be impossible to locate some message destinations. These disruptions can be reduced if indirection is used for referring to message destinations (i.e., if message destinations have IDs other than network addresses).

If indirect reference is used, however, then somewhere in message delivery it must be resolved: the routing function of a subnet needs an address to deliver a message. "Address binding"—the process of discovering the address with which an ID is associated—serves this purpose.

1.4 Name Servers and Dynamic Address Binding

Distributed address binding in communications systems can be performed by name servers, the best known examples of which are the XNS Clearinghouse (Oppen, 1983), and the DARPA Domain-name system (Mockapetris, 1983). Name servers may offer many other services to a communication system, including alias translation, file-name service, user name service, and yellow pages service (i.e., the location of one of several providers of a given service, also known as generic naming). The primary purpose of name servers is to make human interfaces to distributed systems more user friendly. Name servers

operate by use of data bases that record the attributes of various network objects. If subnet addresses are among the attributes that they maintain, then the name servers could be used for dynamic address binding.

Both Clearinghouse and the Domain-name system maintain information on applications-layer objects. The names of these objects are divided into hierarchical components; there are naming authorities of differing hierarchical levels responsible for the name components of differing levels. In both systems, "trees" (i.e., directed, acyclic graphs) are used to model the name space, with namable objects represented by leaf vertices; intermediate nodes represent the hierarchy of naming authorities.

The hierarchical name spaces supported by the Domain-name and Clearinghouse name servers reflect the hierarchical organization of naming authorities. A name server is subordinate to another (its superior) if it is responsible for a "subtree" (i.e., a connected subgraph) of the name space for which the superior is responsible. The hierarchical structure of the name space fixes the order in which servers are queried. If queries cannot be answered because a name is not within a server's purview, then answers are pursued by passing the queries to superior servers.

A name component at one level is unique within the scope of its immediately superior naming authority: there will be no duplicates among the $n + 1$ level components of the names controlled by a level n naming authority. A fully qualified name is globally unique. The Clearinghouse system always uses three levels, corresponding to the "organization," "domain," and "localname" of an object. In the Domain-name system, the number of hierarchical levels in the name space is flexible.

Name server systems are specialized distributed data bases. In these systems, however, almost all accesses are for retrieval. This narrow use allows the usual data base requirements of consistency, atomicity of transactions, and serializability of operations* to be relaxed. The major problems associated with the name server approach to dynamic address binding are:

---

*From the terminology of distributed data bases, an "atomic transaction" is a data manipulation that either completes successfully or leaves the data base as it was before. Simultaneous operations are "serializable" if they leave the data base in a state that is equivalent to one that would result from a sequence, in an arbitrary order, of these operations.

1. Querying the name servers causes delay and overhead traffic.

2. The name servers and the channels leading to them may become performance bottlenecks.

3. If there are few name servers in comparison to network objects, communications may be widely disrupted if a server crashes (a reliability problem).

4. If there are secondary or backup servers, there may be considerable overhead in transferring copies of a master database to them.

5. Existing name server systems do not adapt dynamically to objects undergoing address changes: updates to the attributes of objects, including network addresses, are entered manually by authorized individuals acting on behalf of naming authorities.

6. Current name servers are not adept at bootstrapping—the process of initial system configuration in which user agents find servers and servers find each other. The Clearinghouse relies on broadcasts, which limits it to environments where broadcast is supported (e.g., local area networks). The Domain-name system assumes that each user agent always has access to the address of at least one name server. Another assumption is that each name server at least has access to the addresses of its superior and subordinate name servers. This restricts the Domain-name system to use in environments where server addresses can be expected to remain constant.

7. Name servers are applications-layer entities; they are both inefficient and, because of protocol layering constraints, unsuitable, for translating logical addresses (which are identifiers from the interface between the network and transport layer) to network addresses (which are identifiers from the network layer).

8. For hierarchical name server systems such as Clearinghouse and the Domain-name system, the names of objects have a structure that reflects their assignment to a given naming authority. An object's name must change if it changes domain, which could cause disruptions in communications.

ADB is completely free from problems five through eight, and is less affected than name servers by problems one through four. The next section explains the underlying concepts of ADB, and describes the components and functions of ADB.

2.0 ASSURED DESTINATION BINDING (ADB)

To establish a working vocabulary, this section begins with a brief discussion of a hypothetical problem domain in which ADB is applied; using a specific problem will provide a framework for presenting ADB concepts. The major concepts of ADB are then laid out individually, in order to expose the purpose and motives for the features of the ADB design. We begin by describing ADB in its simplest form, although that form is probably not suitable for use in any communications system. We then discuss embellishments to ADB that would allow it to work effectively for address-binding.

After the major ADB concepts have been presented in piecemeal fashion, there follows a systematic description of the components of ADB, and the functions they must perform. There is also discussion on various design alternatives for achieving these functions, and on issues that should be considered when these functions operate together. Finally, this section closes with a discussion on the behavior of ADB in a changing communications system, and the methods by which an ADB system would adapt to changes in network topology.

2.1 An ADB Problem Domain

We will describe the use of ADB to support address binding in a communications system. In this hypothetical problem domain, users residing on hosts exchange messages by use of a communications system, which can, with some reliability, transfer messages from one network point of access (PA) to another. It is often said that a host's address identifies its location. The concept of a PA now allows the more precise formulation: an address is the identifier of a host's PA (Saltzer, 1982).

We consider a user as merely a potential source or sink of messages; the host on which a user resides is visible to other users of the communications system. We do not propose here to solve the problem of differentiating users on a single host; we assume the communications system has completed its job when a message arrives at its destination host. Yet despite this focus on the problem of translating host IDs to host addresses, our discussion should also suggest the use of ADB as a solution to the problem of locating users or resources on a distributed system.

A class of identifiers used by ADB have unchanging bindings. In our problem, we call these identifiers "host IDs." Host IDs differentiate each host from all others on the communications system. In addition, host IDs are assumed to be well-known within the communications system; they serve as the handles that identify message destinations. This host ID is similar to a DDN X.25 logical address, since it is a low-level identifier that distinguishes one host from another, but does not show a host's location (DCA, 1983).

Another facet of our model is that at any given moment each host on the network has an address. Unlike its host ID, a host's address may change over time. A message source will not necessarily have immediate access to the addresses of other hosts; finding these addresses is the task of ADB.

The term "address" means an element of the "address space"—the set of syntactically valid addresses. The set of addresses in use by a communications system is usually tiny compared with the total address space (e.g., much fewer than one million—perhaps only ten or twenty thousand—of the 3.8 billion syntactically correct DARPA Internet addresses are in use). In light of this, we use the term "current address" to refer to an address that identifies the PA of an online host.

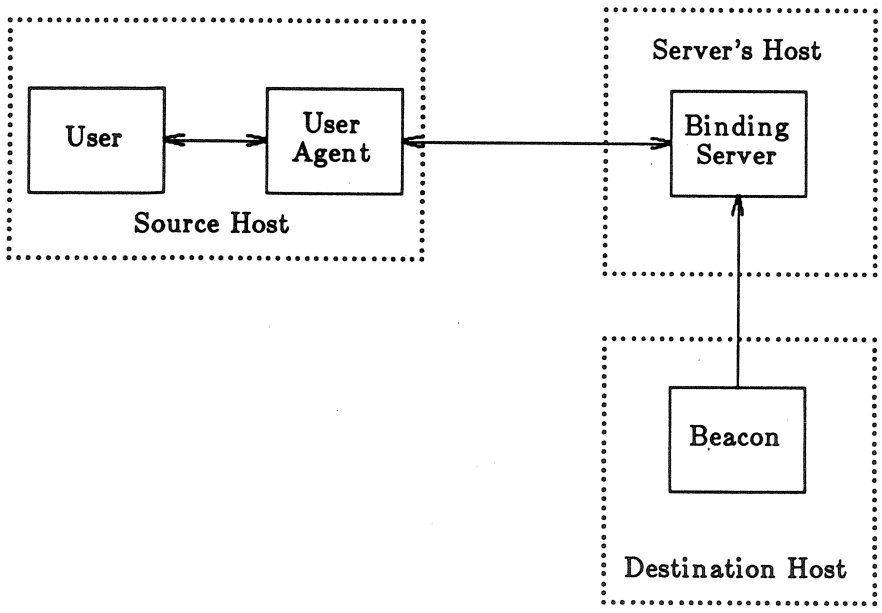
Although the host IDs in our problem are types of names, not all names are IDs, and not all issues associated with naming are pertinent to the subject of address binding. We take assignment of IDs to hosts as given, with the IDs globally unique for all hosts on the communications system. Hence, the administrative issue of name-space management is not addressed in this paper. Also not treated are the local interface problems of either expanding relative names to full system names or translating aliases and abbreviations to IDs. Furthermore, the IDs in our example are for processing by machines. They need not, though they could be, human readable.

2.2 ADB Concepts

Whether simple or embellished, ADB follows the client/server model; that is, user agents query binding servers to obtain bindings for users. Binding servers are able to respond because each host in an ADB system has a "beacon" that transmits its current address binding to the appropriate server. A host will not usually be at the address of the server that caches its binding. User agents, in contrast, are at the address of the

users they represent, and beacons are the address of the hosts whose locations they report. Figure 1 depicts the interactions between a user, user agent, binding server, and beacon.

Figure 1
Interactions Between ADB Components



The beacon transmits its host's address binding to the binding server, from which the user agent obtains it for the user. User and user agent are at the same address.

The fundamental notion of ADB is that, based on a host's ID, beacons and user agents agree about which binding server is appropriate for caching that host's address binding. Beacons and user agents use the same process to transform a host's ID to the address of its binding server.

The following section elaborates on how the current addresses of binding servers are derived from host IDs. We then discuss problems that would arise with using a simple approach to this translation, and offer solutions to these problems.

2.2.1 Naive ADB

We will start with the simplest example of ADB. At each host—or any other addressable network object—we posit a binding server. Each host has a unique, enduring host ID, and an ephemeral current address. The address of the binding server that caches any one host's address binding is derived from that host's ID. The derivation of server addresses from host IDs has two steps (see Figure 2). First, a host ID is transformed into a syntactically correct server address; the result of this transformation will not necessarily be the current address of any binding server. "Assignment" is the name we give to the process of deriving a possibly "fictitious" server address. The second step is to map this assigned value to the most similar current address of a real binding server. "Resolution" is what we call the translation to the address of a real server.

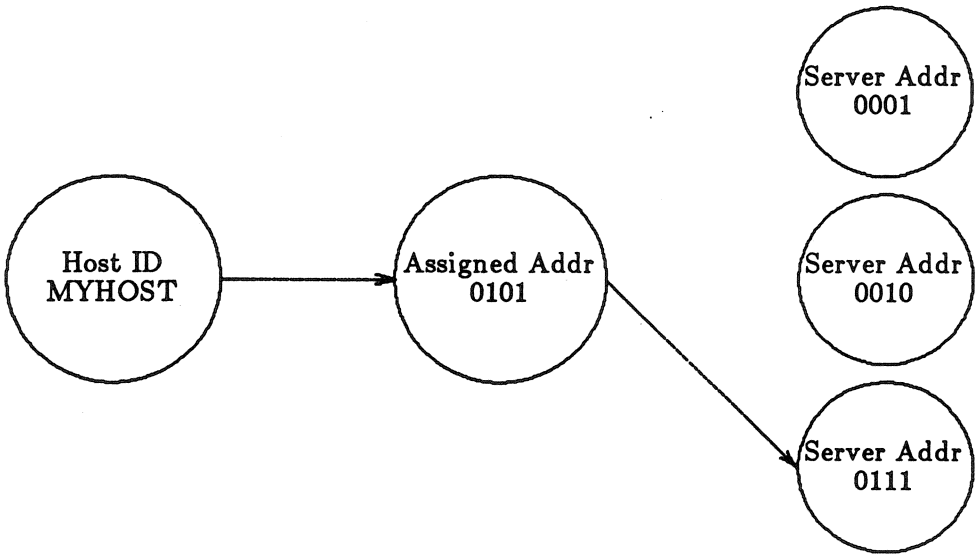
Resolution assures that a host will have a binding server (hence the name, "Assured Destination Binding"). With ADB, the binding server that caches a host's address binding can be located by use of the host's name. Thus, ADB can maintain connectivity across address changes in the communications system.

There are two points to emphasize about transforming a host's ID into the address of its binding server. First, the possibly fictitious address that the assignment function generates from a host ID is as permanent as the host ID itself. Second, a host's binding server will change as current addresses change. If, however, beacons and user agents assign and resolve in the same way, then they should continue to agree about which current address is most like the address that the assignment function generates.

2.2.2 Limiting the Number of Servers

Among the traits that make naive ADB unattractive is its need for a binding server at each current address. The problem, however, of locating binding servers at only a few addresses may seem at first glance to be little more tractable than the original address-binding problem. Nevertheless, we will show that there are a variety of methods for "collecting" the current addresses of binding servers and distributing them among user agents and beacons. Several alternatives for collecting binding server addresses are discussed below, in Section 2.3.3.

Figure 2
Assurance of Finding a Server



In the example above, the current addresses of hosts of ADB servers are 0001, 0010, and 0111. Algorithmic operations convert host ID "MYHOST" to 0101, a "fictitious" address (i.e., there is nothing at that address). The value 0101 is resolved to the actual current address of a server, 0111. So, an address binding for host MYHOST is cached at the server at location 0111.

If there are fewer binding servers than current addresses, then resolution will be modified somewhat: the addresses generated by assignment would be allow to resolve only to the current addresses of servers.

2.2.3 Controlling Server Allocation

There is a potential problem in mapping host IDs directly into address space: it could lead to uneven allocation of hosts to binding servers, resulting in performance bottlenecks. The reason for this concern is that current addresses of communications system are usually highly clustered, because addresses usually have structured formats. For instance, in the DARPA Internet, the first bit of a class-A address indicates address type, and the next seven bits indicate subnet affiliation. Hence, the first eight bits of all class-A addresses of ARPANET hosts are "00001010." Resolution of assigned addresses to current addresses would allocate fewer hosts to the servers whose addresses are within clusters than to outliers.

To ensure that structured address formats do not lead to uneven allocation of the binding server function, we propose that an intermediate level identifier be used. We call this identifier the Virtual Server ID, or "VSID." Functions similar to hash functions could be used to scatter both host IDs and the current addresses of binding servers into the VSID space. Resolution would then be performed to decide which image* from the current addresses of servers is closest to the image of a particular host ID.

Introducing VSIDs would help control how many hosts for which each server caches bindings. For networks not subject to address changes, however, mappings could be devised that send host IDs directly into address space in an even fashion. Or, host IDs could fill the role of VSIDs. Either approach might be incorporated into the design of simplified, efficient ADB systems. Section 2.3.1 explains mapping from IDs and addresses into a VSID space. Section 2.3.4.1 discusses further the problems of fairly distributing loads between ADB servers.

2.2.4 Other Embellishments to ADB

There is another approach that could be of use in balancing the load between binding servers: host bindings could be cached at several locations. In addition to

*From mathematics, the image of a function's argument is the value that the argument is mapped to. Host IDs and current addresses of servers have images in VSID space, since both are mapped to VSIDs.

avoiding the overburdening of any one server, this technique could enhance system reliability, by removing a potential single point of failure in obtaining address bindings.

If hosts cache their bindings at multiple servers, then the assignment process must be modified slightly. Instead of mapping host IDs to single VSIDs (or single, fictitious server addresses, if VSIDs are not used), the elements in the range of the assignment function would be sets of VSIDs (or server addresses). As with mapping from host IDs to single VSIDs, the association between a host ID and its assigned VSIDs would still be fixed; for any one host ID, assignment would always generate the same set of VSIDs.

There are several issues other than performance that are raised by the issue of server allocation. For a variety of reasons, it may be required that only servers of a certain administrative group be allowed to cache the bindings of some communities of hosts. Several approaches to restricting server and host association are discussed in Section 2.4.2.

2.2.5 ADB in Full

The purpose of our address-binding technique is to obtain a host's current address when given its ID, and thereby maintain connectivity across address changes in the communications system. The key components of an ADB system are:

1. Binding servers, which cache address bindings and respond to queries.
2. Beacons (one at each host) which transmit a host's current address binding to one or more binding servers.
3. User agents which query binding servers for users; user agents derive which binding server caches a particular host's binding.

Each binding server has a label—the Virtual Server ID (VSID). A host is linked to the server that caches its address as follows: each host ID is "assigned" a VSID derived from its host ID; the host's address binding is cached at the server whose own VSID is most similar to the host's. "Resolution" is this process of matching the VSIDs of binding servers with the VSIDs derived from host IDs. As mentioned above, to enhance reliability or performance it may be prudent in some environments for a host's address bindings to be cached at several binding servers. If this is done, resolution will yield the VSIDs of several servers.

2.3 Specification of ADB Functions

Whether a beacon is transmitting a binding update or a user agent is transmitting a binding request, the same functions—assignment, resolution, and collection—are used to obtain the current address of a host's assigned binding server. Host VSIDs are derived from host IDs by the assignment transformation. Resolution then maps host VSIDs to server VSIDs. Server VSIDs are in turn bound to server addresses; these bindings are obtained or derived by use of the server address collection.

To separate address-binding issues from routing functions, as well as to suggest ways in which ADB could be installed in existing networks, we will describe ADB as an independent system above the network layer. If entities have the ability to discover their own network addresses, then assignment, resolution, and server address collection could be performed above network layer protocols, by use of network services. As an alternative, however, for future network architectures, ADB functions could be integrated with the routing routines.

2.3.1 VSID Assignment

Assignment maps the set of host IDs into the power set less the empty set* of syntactically correct VSIDs (the range of assignment is not simply the set of VSIDs, because a single host ID may be mapped to several VSIDs). It is crucial to ADB that a given host ID always be mapped to the same set of VSIDs. Hence, the VSIDs assigned to a host do not change, even as servers come and go, and current addresses of hosts and servers change.

There are many ways in which the assignment mapping might be performed, One approach is for each host's assigned set of VSIDs to be well-known by all user agents and beacons; mapping would then be performed by table look-up. This method has the disadvantage of requiring ADB components to acquire and save the VSIDs of all hosts to whom messages might be sent. Saving VSIDs seems redundant, since ADB already requires knowledge of the host IDs of all message destinations.

---

*The "power set" of a set is the collection of all its subsets, including the empty set. For example, the power set of {1, 2, 3} is the collection of sets, {{}, {1}, {2}, {3}, {1,2}, {1,3}, {2,3}, {1,2,3}}.

As an alternative for VSID assignment, a host's VSID might be embedded within its host ID. Here, a host VSID could be readily extracted from the ID of a message destination. This approach would require little overhead, but would restrict the host ID space and could also result in unbalanced demands on server resources (see Section 2.4.1).

A more powerful method of VSID assignment is to derive VSIDs of hosts by performing arithmetic or logical operations on their host IDs. This would cost some computational overhead. No restrictions, however, on the format of host IDs would be entailed. In addition, a well-designed calculation could be integrated with the resolution function to allow VSIDs to be distributed evenly over the set of binding servers.

If a hashing approach is used to map host IDs into the space of VSID sets, then multiplicative hashing would be preferable to modulo hashing. This suggestion stems from the characteristics of these two forms of hashing. In modulo hashing, the initial VSID would be obtained from a hash function, H, by use of the formula:

$$H(id) = id \bmod B,$$

where B is the number of "buckets" (points in the hash function's range).

A property of modulo hash functions is that their arguments will tend to be mapped into clusters if B, the number of buckets, is not prime. For that reason, if B cannot be guaranteed to be prime, it seems better to use the multiplicative approach for hashing (Reingold, 1983):

$$H(id) = \text{floor}(m(id\ \theta \bmod 1)),$$

where m and θ are constants, $0 < \theta < 1$. Subsequent members of a host's set of VSIDs would be obtained by use of a "rehashing" function H', defined as:

$$H'(VSID) = \text{floor}(m'(VSID\ \theta' \bmod 1))$$

The above function takes as its values the integer floor of m (or m') times the fractional portion of the product of **id (VSID)** and θ (θ'). The major limitation of this approach is that **m** and **m'** must contain as many significant digits as the values from the hashing function's range. Likewise, the respective sums of the significant digits of **id** and θ, and **VSID** and θ', should be at least that wide. Hence, implementing the above hash functions may require multiplication of numbers wider than the registers of a computer's ALU, which is a time-consuming prospect.

2.3.2 Resolving Host VSIDs to Server VSIDs

In ADB, resolution is the process that maps a host's VSID to the VSID of a binding server. The most critical requirement of resolution is that all user agents and beacons of a stable ADB system must agree on this mapping for all possible host VSIDs.

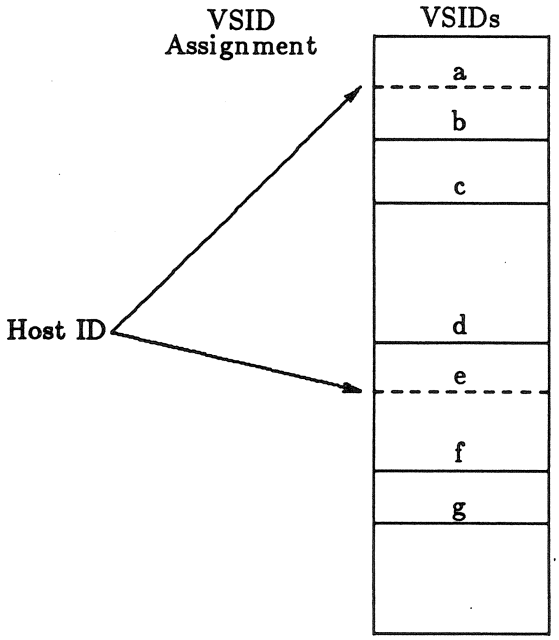
Resolution can be thought of as finding, according to some criterion, which of the server VSIDs is most similar to a host's VSID. For example, a host VSID could be resolved to the server VSID which is its least upper bound (see Figure 3). If the least upper bound is the criterion for resolution, then the space of server VSIDs should be considered circular (i.e., the sequence of server VSIDs "wraps around," so that the lowest server VSID is immediately above the highest).

If there are more hosts than server agents, the function formed by the composition of assignment and resolution will map several host VSIDs to the same server VSID. Also, as the set of servers changes (for example, as a result of sites coming online or going down), a host VSID's resolution—the server VSID to which it is mapped—may change.

Server VSIDs could be bound to server addresses in several ways. One method is to generate server VSIDs algorithmically from their addresses, much as the assignment transformation generates host VSIDs from host IDs. If server VSIDs are generated algorithmically, then ADB will also require the use of an inverse mapping, from server VSID to server address. Figure 4 depicts the mapping of a host's ID to the address of its server, when server VSIDs are generated algorithmically.

If the addresses of servers change often, it could be beneficial to assign VSIDs to servers based on their unchanging host IDs, instead of addresses. Server VSIDs would then be disseminated and collected along with server addresses. The advantage of performing resolution based on server IDs rather than addresses is that server assignment would not change if a server changed address, which could simplify the task of system control. A disadvantage would be that the updates used by communications to collect server addresses would be longer. Resolution based on server IDs is show in Figure 5.

Figure 3
Resolution as Least Upper Bound of VSIDs

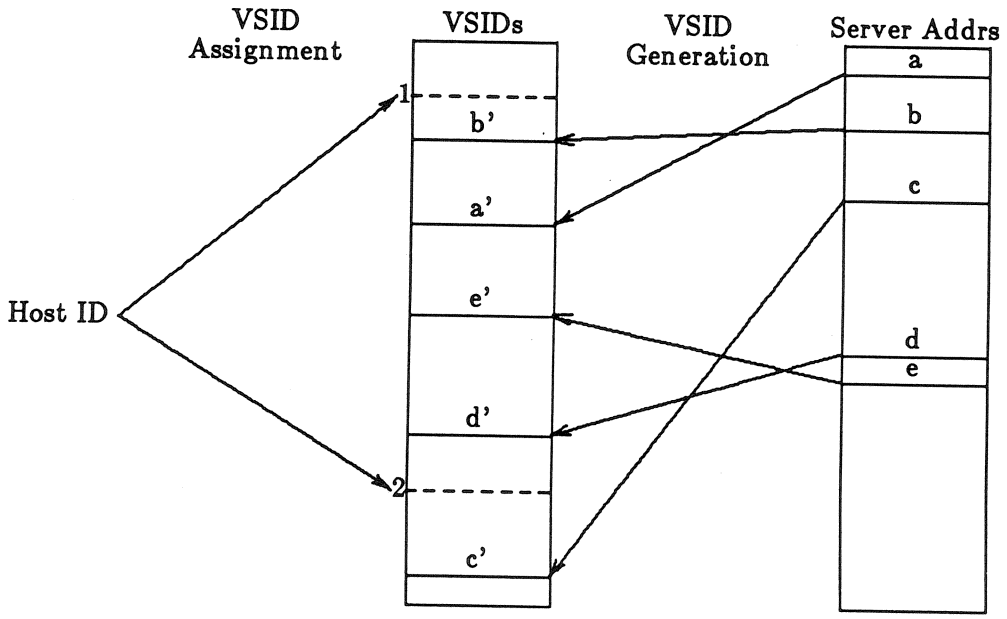


The assignment function associates **Host ID** with VSIDs **a** and **e**. The server VSID that is the least upper bound (lub) of host VSID **e** is **d**. Wrapping around the VSID space, the lub of VSID **a** is **g**. Hence, address bindings for **Host ID** are maintained at a server with VSID **d** and a server with VSID **g**.

2.3.3 Collecting Server VSIDs and Addresses

Any distributed address-binding system faces the problem of bootstrapping—initially finding one or more servers. The mechanism chosen for bootstrapping will depend on the primitive facilities available on the communications system (Birrell, 1982).

Figure 4
ADB with Generated Server VSIDs

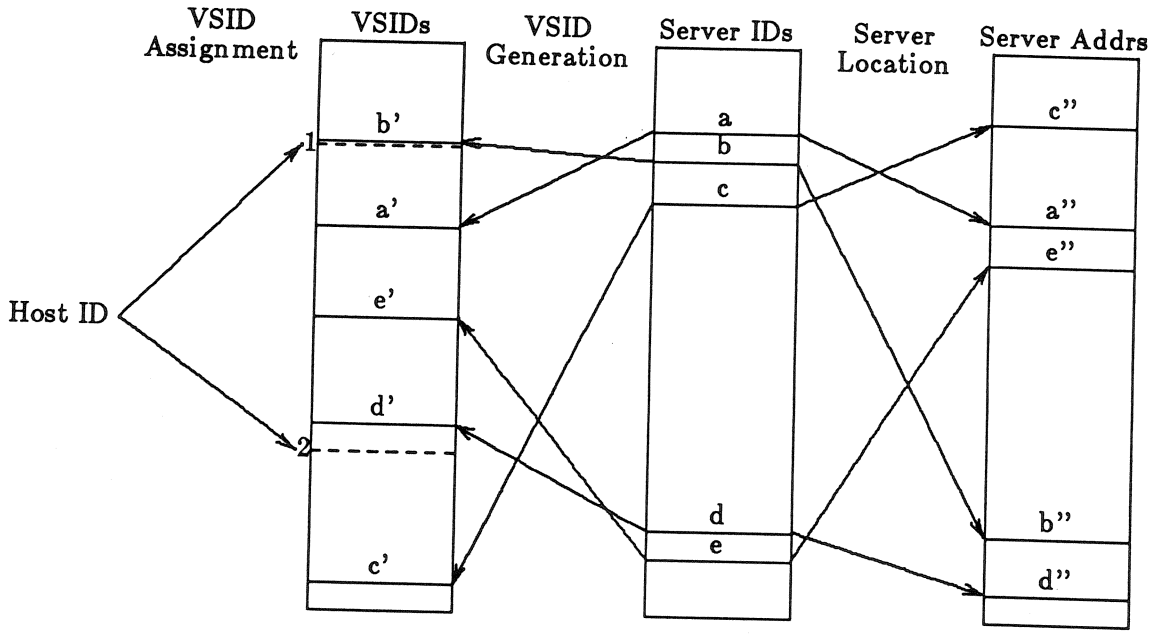


The assignment function maps **Host ID** to VSIDs **1** and **2**. These host VSIDs are resolved into server VSIDs **d'** and **c'**, that were generated from server addresses **d** and **c**. Hence, address bindings for the host denoted by **Host ID** are maintained at servers with addresses **c** and **d**.

In ADB, address collection is used to obtain the initial bindings between the VSIDs and addresses of servers. Also, because server addresses might change and server hosts might go offline, the collection procedure continually updates the list of server VSIDs to which host VSIDs may be resolved (i.e., the range of the resolution function is updated).

The simplest approach to an ADB architecture is for a binding server to be at every PA, and for server VSIDs to be algorithmically generated from server addresses. Collection of server VSIDs addresses would then consist only of obtaining a list of the

Figure 5
Resolution Based on Server IDs



The assignment function maps **Host ID** to VSIDs **1** and **2**. In this example, however, the server IDs, rather than addresses, are mapped into the VSID space. The resolution, by lub, of host VSID **1** is **b'**, the VSID generated from Server ID **b**. The resolution of host VSID **2** is **d'**, the VSID generated from Server ID **d**. Address bindings for the host denoted by **Host ID** are maintained at servers with addresses **b"** and **d"**. Note that though Server IDs and Addresses are clustered, VSID generation spreads the images of server identifiers evenly over the VSID space.

addresses of all functioning PAs. The only special mechanism required to locate servers would be for routing entities of the communications system to pass a list of current addresses to user agents and beacons. Another benefit of completely distributing servers is that the demands on any one server would tend to be light, since a single server would maintain only a few bindings. This approach, however, requires reliable computing and

memory resources for server needs at each current address.

If there are fewer servers than PAs, then a list of current addresses will not suffice for finding the servers. With fewer servers, one approach to finding them is to distribute an authorized list of their addresses, which is in essence the method used by the Domain-name system. This, however, is a poor solution: it is extremely difficult to keep distributed lists consistent, and no current addresses in our hypothetical problem are immune to change.

For more dynamic tracking, servers could announce their locations. This would require a reliable broadcast mechanism, such as flooding. Since user agents and beacons might crash and come up at arbitrary times, each server would be required to broadcast its location with some regularity, regardless of whether its address changed.

Periodic broadcasts by each binding server could cause an excessive amount of overhead traffic. To reduce this overhead, one or more sites could house repositories of server locations. These sites, rather than every server, could periodically broadcast their locations. Servers could then transmit their locations directly to these server directories. Any user agent needing the list of server addresses could then query a server directory. The major problem with the repository approach to collecting server addresses is that the server directories could become performance and reliability bottlenecks.

In a more elegant approach to gathering server addresses, entities of the communications system at a PA would engage their local host—the host that uses the PA for network access—in a dialogue, to ascertain the presence of a binding server. The components of the communications system would then distribute this information along with their routing updates. Once this information had been disseminated, user agents and beacons would obtain the VSIDs and current addresses of servers by querying entities in the communications system.

If a communications system uses a hierarchical address scheme, then the collection of server addresses is more complex. The problem in obtaining address information from the routing entities in a network with hierarchical addressing is that no single routing entity will have routing table entries for all addresses in the system. Instead, network components will be clustered; addresses from remote clusters may even be abstracted as single entries. Outside a cluster, detailed routing information about individual hosts is

not distributed. Instead, information is disseminated as though the cluster were a "super node" in the communications system.

This collapsing of cluster information suggests an approach to server collection in a hierarchical environment: along with a cluster's address, the communications system could distribute its aggregate server capacity and a single VSID—perhaps the minimum VSID within the cluster. Server information could be disseminated with intercluster route updates. Within a cluster, further mappings of host VSIDs could be used to select particular binding servers.

2.4 Integration of ADB Functions

When used together, the ADB functions—assignment, resolution, and collection—must operate efficiently, reliably, and securely. They must also allow for administrative control. One aspect of the efficiency issue is that the operation of ADB functions should not overburden available processor or communications resources. A related issue many implementors may face is devising a means to restrict the allocation of the address-server function to reliable or trustworthy servers. The integrated behavior of ADB functions is described below, and is followed by an enumeration of some approaches for controlling the allocation of hosts to servers.

The performance issue of maintaining stable host allocations (i.e., the issue of avoiding thrashing) is solved by using the least upper bound approach for resolution. With this technique, the crashing of a server causes only the resolutions of its former hosts to change, while the addition of a server changes resolutions only for the hosts whose associated VSIDs resolve to it. For evenly spread allocations, the number of hosts reassigned will be, on the average, the number of hosts divided by the number of servers. Furthermore, since VSIDs are derived from host IDs, a host's address change will not cause it to change servers.

2.4.1 Issues in Allocating Hosts to Servers

There are several independent issues in determining how many hosts each server will cache bindings for. For performance reasons, hosts should be apportioned between servers so that the loading of servers and their neighboring communications resources is acceptable. If a host is assigned to several servers, this problem is compounded: to

determine system loading, the relative frequency with which a host's servers will be queried must be determined. In addition, to satisfy security or administrative policies, it might be necessary to restrict certain communities of hosts to assignment to particular sets of servers.

Fairly distributing the work between servers is a multi-objective optimization problem: total overhead traffic must be minimized, as well as the demands on any one binding server or paths to that server. Even distribution of the server function among PAs is not necessarily optimal. For example, in a network organized in a star configuration, traffic overhead for binding requests and updates would be minimized if a server at the star's central node cached the binding of every host. Note, however, that this could cause prohibitive demands on the resource of the central node.

In addition to topology, traffic patterns will influence performance: some hosts will be popular message sinks, and their respective servers will be highly loaded. Furthermore, since system components will experience differing loads and may have differing capacities, there will be diverse amounts of process and memory resources available for use by binding servers. Host allocation is optimal if it minimizes both contention with data traffic and contention with other computing requirements at nodes.

Keeping work distribution optimized for traffic and available capacity may not be practicable in a distributed system: even if timely enough measurements of loading could be exchanged, the communications overhead could be prohibitive. There are several goals, however, whose pursuit could lead to ADB implementations with adequate performance. Among these is minimizing use of communications resources. Since communications resources tend to be among the first outstripped by demand, this objective should help to drive performance toward acceptable levels. Another goal is to devise a means to ensure that the numbers of hosts assigned to each server remain comparable even as addresses change and servers come online or crash.

2.4.2 Approaches to Host Allocation

A natural solution to restricting host assignment is for the collection procedure to enforce the constraints. The lists of server addresses given to user agents and beacons could differ according to the "community" of a user agent's clients and a beacon's host. User agents, beacons, and servers would need a way of showing their community

affiliations. The problem is more complex if communications between users of differing communities is required. User agents would then need the ability to collect the addresses of servers from these other communities.

An alternative approach to restricting host allocation is to include community indicators with the server addresses. Here, user agents would separate hosts and servers according to community.

Restricting communities of hosts to particular groups of servers would effect a rough form of distributing work between servers. Care, however, in the design of the VSID-to-server mapping offers a less drastic solution to the problem of host apportionment. For example, one means of fairly assigning hosts servers is to randomize the mapping of host IDs to VSIDs, while also mapping the server addresses to the VSID space so that there are roughly equal intervals between address images. The approach of specifying the intervals between VSID images could be modified to achieve other distributions. To assign more hosts to high-capacity servers, for example, the high-capacity servers could be given either greater intervals or multiple points in the VSID space. Either technique would increase the probability of the image of a high capacity server being the least upper bound of a VSID, which would make it likely for high capacity servers to cache a greater portion of bindings.

2.5 ADB in Changing Environments

The elements of an adaptive communications system should have the ability to respond to a variety of changes, such as changes in connectivity, changes in system membership (i.e., hosts crashing or coming online), as well as changes in host addresses. These events are of concern to an address-binding scheme if they can result in failure of bindings to be found or incorrect bindings being disseminated.

In our model, the system components subject to changes are hosts, servers, beacons, and user agents. There is the possibility of a change simultaneously affecting all components at a single PA (e.g., as a result of a packet switching node (PSN) or host crashing). Below we describe the changes of concern to ADB. We then give the relevant issues in detecting changes to system components, and describe the primary mechanism for adapting to changes affecting them, and detail how this mechanism would respond.

Finally, we outline some other approaches that ADB could use to adapt to system changes.

Distributed systems require some amount of time to detect and adapt to a change. Hence, there may be brief periods when the correct address bindings for some hosts are unavailable. We will show that address binding technique could detect and recover within finite time from system changes that could affect address-binding. The speed with which implementations must recover is determined by user requirements and by the rate at which perturbations occur.

2.5.1 Required Adaptiveness

For ADB to work, beacons must detect births and address changes of their local hosts. On the other hand, detecting the death of a host is not necessary for ADB; users may need the address of a host that is down or offline. Hence, in normal operation, we envision a beacon transmitting binding updates, if possible, whenever its host comes online.

Residing at hosts, user agents can come online, and die. No explicit detection for the birth of user agents is necessary; all that is required is that a user agent begin the transmission of binding updates to the servers to which its local users are assigned.

It is debatable whether a failed user agent must be detected by other ADB components. For the most part, failure of a user agent only disrupts its own users: users at the address of a failed user agent would have no means of obtaining address bindings, and hence could not transmit to new destinations or to hosts that had changed addresses. The actions that could be taken to respond to a failed user agent (i.e., move its users, or activate a backup agent) should be under local control. Compounding the problem of remote responses is the lack of mechanisms that remote systems could use to request that a user agent be replaced. Hence, our address-binding technique does not specify a response to make to the failure of a user agent.

By a similar argument, the failure of a beacon is also a local issue. The failure of a single beacon has limited, mostly local consequences: if a beacon is not operating, the current bindings of its host may be unobtainable by message sources, so it may be denied incoming messages. As with user agents, there is the problem of a lack of distributed

mechanisms for coping remotely with beacon failure. Therefore, our address-binding technique does not specify a response to make in the event that a beacon fails.

In contrast, ADB must adapt to changes in the status of servers. The failure of a server could impede address binding, resulting in the eventual isolation of a set of users. This is because a dead server would quietly absorb all binding requests and updates sent to it. Furthermore, the birth or death of a server will affect the process of resolving VSIDs. However, there are actions that beacons and user agents can take to detect and mitigate the effects of the birth, death, or address change of a server.

Several of the means ADB uses to detect change are implicit in its functional requirements. For example, a beacon must be able to obtain its current address and detect its changes. Detection of changes in server status is also closely related to the functional requirements of the address-binding scheme. To resolve VSIDs into server addresses, each user agent must be able to collect the current addresses of all active servers. As will be seen, adaptation to changes in server status requires only that user agents and beacons use an updated list of current addresses of servers when resolving host VSIDs.

2.5.2 Adaptation Mechanisms

ADB adapts to system changes primarily through its two fundamental processes:

1. Beacons sending binding updates to their hosts' respective servers.
2. User agents performing collection of server addresses.

Periodic execution of these functions is an immediate solution to the problems of address changes and the introduction or failure of binding servers. Both functions could be performed in either timer-driven or event-driven fashion. The timer-driven approach would tend to have slower response, and would impose a high communications overhead. However, the timer-driven approach would simplify the host-monitoring task of beacons.

The transmission of binding updates to servers copes with changes affecting hosts. A beacon will detect changes in the address of the local PA. Either upon detection of a change in status or at the expiration of a timer, the beacon transmits a binding update to the servers denoted by its hosts' VSIDs. As soon as the servers denoted by the VSIDs have received binding updates, they will be able to respond correctly to binding queries.

Server status could be monitored by the mechanism for collecting server addresses, since the collection procedure assumes that news of a server birth, death, or address change can be propagated throughout the communications system. The normal operation of resolution would then reallocate hosts to new servers. Hence, since binding updates are regularly transmitted, binding updates would soon arrive at a host's newly assigned server. Once this occurs, the address-binding service will be restored.

3.0 ISSUES FOR FURTHER STUDY

Our outline of ADB suggests many topics for additional study. Certainly the functions and characteristics of ADB could be investigated in greater depth. Another area for future work would be to specify ADB at the level of detail required for its use in a communications system. In addition, a more thorough exploration of the possible applications of ADB to other problem domains could be rewarding.

Before fielding ADB, alternatives for implementing the assignment, resolution, and collection procedures should be studied. These functions should be computationally efficient, yet must also allocate server resources in a nondetrimental fashion. Further guidelines for developing such functions for given ID, VSID, address spaces, and available network services would be useful.

The issues of integrity and trust within an ADB system should also be studied. Particularly interesting are methods of authentication of binding updates, and also methods for restricting participation in address binding to trusted entities.

For ADB systems in which a host's bindings are cached at several servers, there is a further interesting topic: determining the means by which a user agent might select optimal or near optimal servers for queries. The solution to this problem, however, will no doubt depend on the particular communications system that uses ADB.

In addition to deeper investigation of the ADB functions, there are several areas in which the actions of ADB might be expanded. For example, if hosts are highly mobile, it is possible that ADB performance could be improved by predicting host locations: anticipatory binding updates might be forwarded to servers. Clearly, this would require complex recovery mechanisms (e.g., what if a bad guess is made about the host's path and velocity?), but it could improve the ability to reduce delay and maintain connectivity.

Another expansion of ADB could be effected in systems with binding servers at each PA. In these systems, ADB could be integrated with message delivery. The destination VSID of every arriving message could be checked; if it does not match the VSID of the location at which it has arrived, then ADB could be used to forward the message to its proper destination. The benefit of this scheme would be that the beacon of a host whose

address had changed could send binding updates to its previous address; the effect would be to leave a trail of correcting bindings.

In using ADB for message forwarding, the regular binding server of a destination could be the last resort for forwarding messages. Hence, forwarding trails could be disposed of after some time, without fear of disconnecting the network. This avoids the major problem that has been cited for using a forwarding trail to update bindings (Powell, 1983).

Other further work in ADB would be to undertake a detailed specification for its use within a communications system. ADB could be specified as an integral component of a newly designed communications system; no doubt this approach offers the best performance and versatility. Yet this is not the only possibility for fruitful use of ADB in a communications system. For example, one might use ADB to introduce logical addressing within a communications system, thus increasing the system's robustness.

The concepts of ADB might also be applied to areas other than address binding. For example, it seems feasible to use ADB to translate host aliases into unique IDs; VSIDs could be assigned to aliases, and the ID-to-alias bindings cached at servers denoted by these VSIDs. ADB might also be used to support generic naming for some services, in which the ability to request the first available server would be an asset. To use ADB in this fashion, it could be beneficial to have the binding servers also assume the function of resource allocators. In an even more general use of ADB for resource allocation, ADB could control access to resources and remote procedures in a distributed computer system.

Another interesting extension of ADB would be using it to provide a means of decentralized public key distribution. In essence, ADB defines a means by which queries for information can rendezvous with answers. It would seem straightforward for a "key server" to be designed much as this document describes a binding server.

4.0 CONCLUSIONS

The distributed address-binding system we have described has several points in its favor:

1. Other than uniqueness, it places no constraints on host IDs.
2. It is inherently robust and adaptive.
3. By support of logical addressing, it offers a solution to the mobile host and multihoming problems.
4. It is free of many of the disadvantages of distributed database name-server systems.

ADB neither prescribes nor precludes any structure to the IDs of hosts. Hence, implementors are free to design a name space for their own needs, or use existing values for host IDs. An added advantage to the flat ID space of our scheme is that the recursive searches typical of hierarchical name spaces could be avoided. In a hierarchical address-binding scheme such as the Domain-name system, a name server might be required to query other name servers, and so on, to satisfy a binding request. This need for multiple server accesses could result in lowering performance of the address binding (Lantz, 1985). In ADB, however, user agents would always transmit binding requests directly to the appropriate server.

ADB is also inherently fault tolerant. Because several servers may cache a host's address binding, and because resolution reassigns hosts to healthy servers if their servers fail, server crashes will not always preclude the servicing of binding requests. ADB also supports other tactics, such as migration, for achieving robustness. Given enough warning, moving hosts to differing PAs can lessen the loss of service caused by sites crashing. A primary difficulty in moving the components of a distributed system, however, is the correction of the bindings (Lau, 1984). ADB allows users to maintain connectivity across address changes caused by host migration.

A further advantage of ADB is that it could support the dynamic resolution of logical addresses to current routing addresses. This capability offers a solution to the problem of mobile hosts connecting to a terrestrial network. In a nutshell, the mobile

host problem is nothing more than the possibility that a host's address might repeatedly change. Clearly, ADB is well suited for solving this problem.

The use of logical addresses also provides a consistent mechanism for handling multihomed hosts. Multihomed hosts have at least two network addresses on the same communications system. Some multihomed hosts use each address for only certain classes of traffic, while others place no such restrictions on the use of their interfaces. In existing networks, directing traffic to the correct interface of a multihomed host requires specific local knowledge about that host. Logical addressing could give a systematic means for distributing this information. If a host's interfaces are restricted, then they might be given distinct logical names; a single host would then appear to be multiple network objects. On the other hand, if the interfaces are not restricted, then a beacon process might send one or both addresses to all its host's binding servers. The endpoints of connections could then be defined by logical addresses, which would allow end-to-end connections to continue even one of a host's network interfaces fail.

Because the address binding service of our system is not limited to a few sites, many of the disadvantages usually associated with name-server systems could be avoided. For instance, if many servers participated in an address-binding system, each with a small purview, then high-volume data transfers typical of name servers would not be required. Also, use of many limited servers would render ADB less vulnerable than the distributed database approach to widespread loss of service as a result of a single component failure. Also, congestion at the points of service would be avoided.

4.1 Summary

ADB is an adaptive, robust, fault-tolerant technique for locating objects in a distributed system. The components and functions required by ADB are described in the context of the use of ADB to perform address binding for a communications system. ADB would adapt to events such as changes in the addresses of network objects, failure of switching nodes, and failure of individual ADB components. The most efficient use of ADB would be to integrate it with the routing routines of a communications system. It seems possible, however, to implement ADB as a system that uses the services of an existing network.

ADB could be used for translating logical addresses to addresses; it seems much better suited for this task than name server systems. ADB's support for logical addressing in turn offers a solution to both the mobile host and the multihomed addressing problems.

Although detailed development of ADB concepts is not complete, it appears to be useful in a wide range of applications. For instance, ADB could distribute cryptographic keys in a public key system. With some enhancements, ADB could be used to allocate resources and procedures in a remote procedure call environment. More generally, ADB may be used in a distributed system to disseminate and gather information that can be bound to an identifying label.

REFERENCES

Birrell, Andrew D., Levin, Roy, Needham, Roger M., and Schroeder, Michael D. (April 1982), "Grapevine: An Exercise in Distributed Computing," *Communications of the ACM*, 25:4, pp. 260-274.

DCA, (December 1983), *Defense Data Network X.25 Host Interface Specification,* Defense Communications Agency, DDN Program Management Office.

Lantz, Keith A., Edighoffer, Judy L., and Hitson, Bruce L. (August 1985), "Towards a Universal Directory Service," *Fourth Symposium on the Principles of Distributed Computing.*

Lau, Francis C.M. and Manning, Eric G. (Oct 1984), "Cluster-Based Naming for Reliable Distributed Systems," *Proceedings of the Fourth Symposium on Reliability in Distributed Software and Database Systems*, pp. 146-154, Bethesda, MD.

Mockapetris, Paul (November 1983), *Domain Names - Implementation and Specification,* USC Information Sciences Institute.

Oppen, Derek C. and Dalal, Yogen K. (July 1983), "The Clearinghouse: A Decentralized Agent for Locating Named Objects in a Distributed Environment," *ACM Transactions on Office Information Systems*, 1:3, pp. 230-253.

Powell, Michael L. and Miller, Barton P. (Oct 1983), "Process Migration in DEMOS/MP," *Proceedings of the Ninth ACM Symposium on Operating System Principles*, pp. 110-119.

Reingold, Edward and Hansen, Wilfred (1983), *Data Structures,* pp. 332-339, Little, Brown, & Co..

Saltzer, Jerome H. (1982), "On the Naming and Binding of Network Destinations," *Local Computer Networks*, pp. 311-317, North-Holland Publishing Company.

Shoch, John F. (Fall 1978), "Inter-Network Naming, Addressing, and Routing," *IEEE Proc. COMPCON*, pp. 72-79.

Su, Zaw-Sing and Mathis, James E. (Oct-Nov, 1984), "Internetwork Accommodation of Network Dynamics: Naming and Addressing," *Proceedings of the Seventh ICCC*, pp. 681-685.

Sunshine, Carl A. (April 1981), "Addressing Problems in Multi-network Systems," *IEN 178.*

GLOSSARY OF ACRONYMS

ADB	Assured Destination Binding; a self-configuring dynamic binding technique
ALU	Arithmetic Logic Unit
ARPANET	Advanced Research Projects Agency Network
DARPA	Defense Advanced Research Project Agency
DCA/DCSDS	Defense Communications Agency/Defense Communications System Data System
DDN	Defense Data Network
PA	Point of Access; an interface to a network
PSN	Packet Switch Node
TCP	Transmission Control Protocol (MIL-STD 1778)
VSID	Virtual Service ID; an identifier used to associate hosts with binding servers
XNS	Xerox Network Systems

Tsuchiya, Paul F. (September 1987), "The Landmark Hierarchy: Description and Analysis," MTR-87W00152, The MITRE Corporation.

DISTRIBUTION LIST

MITRE Washington

A-10 G. MacDonald
A. Tachmindji

D-14 J. Babcock
E. Brady
J. Quilty
A. Roberts

W-30 G. Benke
R. Britton
R. Grayson
L. Keane
B. Moran
R. Shirey
J. Slaybaugh
L. Wentz
D. Woodall
M. Zobrak

W-31 S. Bhanji
H. Duffield
D. Gomberg
R. Hansen
D. Hartmann
W. Lazear
T. Louden
S. McLeod-Reisig
M. Meltzer
A. Messeh
R. Miller
R. Stine (10)
P. Tsuchiya (10)
A. Whitaker
R. Wilmer
D. Wood (2)
Associate Tech. Staff
Technical Staff

W-34 T. Minton
E. Schmidt
A. Schoka

W-35 C. Bowen
D. Jurenko
W. Kinzinger
W. Stewart

W-36 E. Boyle
B. Brooks
G. Lipsey
D. Zugby

W-37 R. Haller
R. Pesci

W-75 W. Blankertz

MITRE Washington Library

Record Resources (2)

MITRE Bedford

A-10 C. Zraket

D-110 G. Koehr
I. Richer

D-111 J. Woodward

D-114 H. Bayard

D-118 S. Ames

MITRE Bedford Library

External

SPONSORS

DCA/DCSO

Mr. W. Harding, Code B101
COL T. Herrick, Code B600
Mr. E. Schonborn, Code B601

DISTRIBUTION LIST (Concluded)

Mr. L. Tabacchi, Code B602
LtCol G. Mundy, Code B602A
Mr. W. Grindle, Code B610
Capt M. St. John, Code B612
Mr. J. Powell, Code B620
LtCol C. Holland, Code B630
LTC D. Schreiner, Code B640
Mr. J. Milton, Code B650

OSD/C^{3}I

Mr. M. Corrigan

DARPA

Dr. D. Perry

UNCLASSIFIED
SECURITY CLASSIFICATION OF THIS PAGE

REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION (U)	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY N/A	3. DISTRIBUTION / AVAILABILITY OF REPORT
2b. DECLASSIFICATION / DOWNGRADING SCHEDULE N/A	
4. PERFORMING ORGANIZATION REPORT NUMBER(S) MTR-87W00050	5. MONITORING ORGANIZATION REPORT NUMBER(S)

6a. NAME OF PERFORMING ORGANIZATION The MITRE Corporation	6b. OFFICE SYMBOL *(If applicable)* W31	7a. NAME OF MONITORING ORGANIZATION
6c. ADDRESS *(City, State, and ZIP Code)* 7525 Colshire Drive McLean, VA 22102		7b. ADDRESS *(City, State, and ZIP Code)*
8a. NAME OF FUNDING / SPONSORING ORGANIZATION DCA	8b. OFFICE SYMBOL *(If applicable)* B600	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS *(City, State, and ZIP Code)* 8th and South Courthouse Rd. Arlington, VA 22204-2199		10. SOURCE OF FUNDING NUMBERS

PROGRAM ELEMENT NO.	PROJECT NO.	TASK NO.	WORK UNIT ACCESSION NO.

11. TITLE *(Include Security Classification)*

Assured Destination Binding: A Technique for Dynamic Address Binding (U)

12. PERSONAL AUTHOR(S)
Robert H. Stine, Jr. and Paul Tsuchiya

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT *(Year, Month, Day)*	15. PAGE COUNT
Final	FROM _____ TO _____	1987 March	35

16. SUPPLEMENTARY NOTATION

17. COSATI CODES			18. SUBJECT TERMS *(Continue on reverse if necessary and identify by block number)*
FIELD	GROUP	SUB-GROUP	

19. ABSTRACT *(Continue on reverse if necessary and identify by block number)*

Assured Destination Binding (ADB) is an approach to dynamic binding that could be used within a distributed system to disseminate and gather any information that can be bound to an identifying label. ADB has potential applications in many areas, including the distribution of public cryptographic keys and the locating of resources in a distributed system. The primary application expected for ADB, however, is address binding - the translation of names into addresses. Because ADB may be used to effect logical addressing, it also offers a solution for the mobile host problem and multiport network problems.

This report describes the components and functions of an ADB system used for address binding in a communications system. Because of the wide potential for applications of ADB, a general description of ADB's underlying concepts and major design alternatives is also presented.

20. DISTRIBUTION / AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☐ UNCLASSIFIED/UNLIMITED ☐ SAME AS RPT. ☐ DTIC USERS		
22a. NAME OF RESPONSIBLE INDIVIDUAL Mr. J. Powell	22b. TELEPHONE *(Include Area Code)* 285-5101	22c. OFFICE SYMBOL B620

DD FORM 1473, 84 MAR

83 APR edition may be used until exhausted.
All other editions are obsolete.