Cross-Substrate Advertisement: Building Overlay Networks for Heterogeneous Environments

by

Majid Valipour

A thesis submitted in conformity with the requirements
for the degree of Master of Applied Science
Graduate Department of Electrical and Computer Engineering
University of Toronto

Abstract

Cross-Substrate Advertisement: Building Overlay Networks for Heterogeneous Environments

Majid Valipour

Master of Applied Science

Graduate Department of Electrical and Computer Engineering

University of Toronto

2010

Self-organizing overlay networks have emerged as a powerful paradigm for building networks and providing network services. Unlike traditional infrastructure-based networks, self-organizing overlay networks are formed and operated in a fully distributed manner, and do not rely on centralized mechanisms for network formation, management, or operation. Most approaches to overlay networks assume universal access to the Internet where the Internet forms a single global substrate network over which an overlay network can be built. In this thesis, we consider the construction of overlay networks over multiple heterogeneous substrate networks, where any data link, network layer, or even overlay network can constitute a separate substrate network. Such networks can support seamless interconnection of mobile nodes without access to a network infrastructure to nodes connected to an infrastructure. Application areas are wireless community networks, vehicular ad hoc networks, and sensor networks. We present the design, implementation and evaluation of Cross-Substrate Advertisement (CSA) mechanisms that enable and facilitate the formation of overlay networks in a heterogeneous environment of multiple substrate networks. A key difficulty of CSA is to address the problems that arise from the more complex address bindings, where a single logical identifier is bound to multiple substrate addresses. We present two principal mechanisms for exchanging information on address bindings, i.e., direct address exchange and third-party address exchange, and

evaluate their effectiveness in different types of substrate networks. The CSA mechanisms have been implemented in the HyperCast overlay protocol architecture. We have conducted measurement experiments of the CSA mechanisms on a local Emulab testbed. The experiments show that our CSA methods are effective in disseminating address information in large networks and are robust in the presence of network disruptions. We conclude that the proposed CSA mechanism can improve the connectivity and stability of overlay networks in heterogeneous environments without incurring a significant overhead.

Acknowledgements

First I would like to thank my supervisor, Jörg Liebeherr, for his continued support and trust in me. He has been a great mentor whose expertise, understanding and insight were an invaluable help in my graduate studies.

I deeply enjoyed the stimulating discussions I had with my fellow members in the Network Research Lab over the years, and want to thank all past and present members for their help and support. I would especially like to thank Yashar Ghiassi, Tony Zhao and Jon Lei for their help and friendship.

Finally, I am forever in debt to my family for their love and support. I would like to express my deepest gratitude to my parents, Nahid Seilanizadeh and Hamid Valipour, who have always been supportive of my academic success.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Over the last four decades, the Internet has transformed how people interact, how businesses are run, and how science is practiced. A large number of applications rely on the Internet for their communication needs. The growth of its users and applications has brought new stresses and unforeseen requirements which have revealed numerous limitations in the Internet architecture. There is a growing consensus in the networking community that some aspects of the Internet design, based on decisions made in the 1970's, is severely limiting its security, availability, flexibility, and manageability. Authors of [13, 15, 3] suggest that such limitations cannot be addressed with incremental adjustment within the current architecture and a fundamental rethinking of the Internet (a clean-slate design) is required.

Today, a diverse set of communication methods with different medium, technology, and protocols are used to connect a diverse set of devices, ranging from powerful personal computers to power and computation constrained sensors. While the underlying communication technologies of the Internet have continued to evolved overtime, its fundamental design principles have remained more of less the same. The Internet provides a simple best-effort packet-switched delivery service, one fixed-size address per network interface. Any other functionality e.g., a reliable stream service, is implemented at the edge.

The authors of [27] argue that elements of the Internet design, in particular the end-to-end principle, have hindered innovation in the Internet architecture. Desired mechanisms such as multicast, QoS, and security have not been integrated except on a limited scale, i.e., in enterprise network that are isolated from the public network.

Overlay networks have been used as a mean to deploy new services and to enable innovation. More recently, there have been proposals that take advantage of overlay networks to foster heterogeneity in the Internet architecture.

1.1 Overlay Networking

An overlay network is a network which is built on top of other underlay networks (*substrate* network) for the purpose of providing a service that is not already available in the substrate network. Nodes communicate using the substrate network to organize themselves into a topology which is the overlay network. In an application layer overlay network, the overlay nodes are running on Internet end-hosts and these instances are connected using transport layer links also known as *logical link*s.

Application layer overlay networks are an attractive method for the deployment of new services in the Internet without requiring any changes in the underlying IP protocol. These networks provide previously non-existent services. Overlay networks are often self-organizing to reduce management costs. Similarly, overlay networks are often decentralized to improve the scalability and robustness.

Recently, overlay networks are being considered as one of the enabling technologies for the next generation network architecture [3, 9]. In these proposals, overlay networks are employed as a method of deploying new services on routers and in the middle of the network, a role played by the IP protocol in the Internet architecture. For example in [3], virtualization has been used to divide links and routers into slices to support multiple experimental overlay networks simultaneously. These proposals change the role

of overlay networks in the overall architecture. Often, overlay networks are understood in the Internet architecture as another layer between the application and the Internet. Recent proposals such as [3] argue that overlay networks are well capable of playing the role that the Internet Protocol (IP) used to play in the Internet.

There are many issues that have to be resolved before these proposals can be realized. Some issues arise due to the heterogeneity of underlying networks at level below the network layer. This creates new problems and challenges not present in application layer overlay networks.

1.2 Network Heterogeneity

Network heterogeneity manifests itself in terms of diversity in the communication technologies, as well as in the types of devices that these networks interconnect. Today, networks connect devices whose power, processing, and communication capabilities greatly differ from each other. According to [34], the heterogeneity of networked devices is more likely to increase in the future to accommodate emerging applications such as smart environments, factory automation, surveillance, vehicular communication, environmental and biomedical monitoring and others. As a consequence, we will see an increased diversity in communication mediums and data link layer protocols.

Another recent development is the introduction of network-enabled devices that are capable of using multiple methods of communication. For example, today most notebooks are capable of exchanging information over WiFi (802.11 b/g/n), Bluetooth, Ethernet, and telephone lines via a modem. Furthermore, smartphones are capable of using WiFi, Bluetooth, and third generation cellular networks. Availability of these multi-interface devices capable of accessing multiple networks requires inter-networking mechanisms to build networks that span multiple different substrate networks.

1.3 Overlay Networks and Underlay Heterogeneity

This thesis addresses the design and implementation of mechanisms that can assist overlay networks to deal with heterogeneous substrate networks. In particular, the problem that arise due to one-to-many address bindings that exist in this setting. We design components for an existing overlay network system (HyperCast) to enable overlay networks to operate over multiple heterogeneous substrate networks. We show that using our solution and with minimal modifications, existing overlay protocols can operate across different substrates and provide connectivity among nodes in interconnected heterogeneous networks. Our efforts are a step toward a network architecture that embraces heterogeneity as a core principle.

1.4 An Overlay-based Network Architecture

While today's Internet provides a single network that supports all types of electronic communications, we envision that next-generation network designs will be built on specialized networks that serve individual applications. The core functions of each network such as forwarding, naming and addressing, security, management and other will be designed and implemented to meet specific requirement of the application it serves. A large number of these networks is expected to coexist and serve radically different purposes.

These networks will be deployed as overlay networks on top of existing lower layer networks. Each overlay network will be primarily constructed by users of its application. Infrastructure nodes will be requested to provide connectivity between different substrates. These infrastructure nodes will join the overlay network in a self-organizing and on-demand fashion.

For our vision to be realized, many challenges have to be overcome. There are issues regarding security, resource allocation and management, economical viability and incentives, network heterogeneity, self-organization and discovery mechanisms. In this

thesis we provide mechanisms that allow overlay networks to handle multiplicity and heterogeneity of substrate networks.

We use an address binding model that works in a environment with multiple heterogeneous substrate networks. In this address model, the identifier of the overlay node is bound to multiple substrate addresses, one address for each substrate that the overlay node is connected to. *Cross-Substrate Advertisement* (CSA) is the mechanism that helps overlay protocols to implement this new model. CSA uses an address format that can be carried across different substrate networks. CSA provides different methods for exchanging address bindings between nodes in the overlay network.

1.5 Thesis Organization

Chapter 2 discusses related research and previous work on this subject. Chapter 3 discusses concepts of overlay networking and how they are affected by existence of heterogeneous substrate networks. Our discussion in this chapter is in the context of the HyperCast project, and we use abstractions and primitives that this project defines. Chapter 4 presents the design of cross-substrate advertisement, a mechanism that allows overlay protocols to operate on multiple heterogeneous substrates with minimal modifications. Implementation of CSA and its address exchange methods are outlined in Chapter 5. Chapter 6 presents the design of experiments to evaluate our design and implementation and analysis of their result. Finally, Chapter 7 concludes the thesis and outlines future works.

Chapter 2

Previous Work

In this chapter we discuss the related work. The first section investigates the current state of overlay networking. The following sections explore the literature on approaches for inter-connecting heterogeneous networks. The last section summarizes and compares previously discussed solutions.

2.1 Overlay Networks

Overlay networks have been extensively used as a method of providing new services in the Internet. Examples of such services are multicast [8, 24], quality-of-service [36], robust routing [2], and content-distribution [1]. In this context, overlay networks are application-layer networks, meaning that nodes in the overlay networks are applications that run on the end-hosts which are connected to the Internet. Nodes in these networks self-organize the network topology, and communicate in a peer-to-peer fashion. Application-layer overlay networks generally rely on the Internet to provide end-to-end connectivity.

Network-layer overlay networks have been used as means to provide end-to-end connectivity among nodes in different data link layer networks by inter-connecting these networks. An example of such overlay networks is the Internet which interconnects multiple data link networks [10]. In Delay Tolerant Networking (DTN) [14], an overlay

network is constructed to interconnect nodes that exist in networks with different timing characteristics and potentially different forwarding processes. Two main parts that overlay networks of this kind have to provide are a forwarding process and an addressing scheme.

2.2 Overlay Networks and Identifier/Locator Separation

One of the most attractive features of overlay networks is their ability to decouple the identities of nodes (*identifier*) from the addresses of their attachment points in the substrate networks (*locators*). This feature simplifies support of privacy and mobility. An identifier which is independent from the location in the substrate network need not be modified when the node changes its network attachment point. This makes it possible to achieve mobility and multi-homing with respect to the underlay substrates without the additional complexity needed for communicating new or additional identifier(s). For this address decoupling to work, each node needs to maintain a list of bindings between the identifier of its neighbors and their addresses in substrate networks.

An overlay network defines the address space from which node identifiers are selected. Example of identifiers used by overlay protocols are self-authentic cryptographic identifiers [5, 30], coordinates in multidimensional plane [24, 32], or random identifiers from a flat space [33, 35] (common among DHTs). In many *structured* peer-to-peer overlay networks, identifiers have significant topological relevance. In fact, in such cases the node identifier is also its *locator in the overlay topology* which should not be confused with its locators in substrate networks.

The idea for separating the identifier and the locator has previously been explored in context of alternative inter-networking protocols such as PIP [16], Nimrod [7], and 8x8 (also known as GSE) [31]. In 1993, PIP pioneered the idea of separating identifier from

locators by introducing the concept of forwarding directives and hierarchical identifiers composed of multiple flat IDs. Later this work became the basis for IPNL (IP Next layer)[17] which uses fully qualified domain names (FQDN) as identifiers and IP addresses as locators.

Similarly, many next-generation network architectures such as FARA [11], Triad [18], SNF [20], IPNL [17] and 4+4 [37] rely on naming structures that separate identifiers and locators. These naming and addressing systems are significantly more flexible, expressive, and comprehensive than the hierarchical IP address space.

In FARA, each node maintains a local set of bindings (associations) between node identifiers and their locator (i.e., forwarding directives). These bindings are purely end-to-end and are invisible to routers. FARA does not implement any particular protocol but instead tries to demonstrate that identifier/locator split does not prevent design of efficient protocols.

When simply separating a node identifier from its location, it becomes trivial to spoof a host identity and hijack its packets from anywhere in the network. To avoid this, it is necessary to use cryptographic methods for verification of identities to prevent simple spoofing of the identity. An elegant implementation of this approach has been employed by the Host Identity Payload (HIP) [30]. HIP uses a cryptographic key to represent the identity of a node. A hash of this key is carried in a shim layer between IP and transport layer headers to which application-layer entities bind. Only the owner of the key can cryptographically prove that it is the legitimate owner of this hash. This way, HIP separates identity and location while avoiding simple identity spoofing. Although HIP was originally designed as an extension of the IP network but its approach has been adopted by many proposals that advocate the separation of the identity and location that are not based on the IP network.

An *overloaded* address[1] provides a much simpler solution for this kind of hijack at-

[1]An address that is both identifier and locator, e.g., an IP address in the Internet architecture.

tacks. Since the forwarding process guarantees that the packet will always be delivered to the host located at the address, assuming routers along the path have not been compromised, one can safely conclude that the packet is as well delivered to the host identified by the address.

2.3 Network Heterogeneity

The homogeneous internetworking layer of the Internet allows both applications and underlying communication protocols to be developed independently [6, 10]. This has been cited as a major reason for the success of the Internet. Recently, however, it has been seen that due to this homogeneity, the IP protocol suite can make any fundamental change difficult [12]. Furthermore, the packet-switched, best-effort IP network is inherently incapable of properly utilizing some of the recently developed communication technologies, and devices [29]. For example, complexity and overhead added by IP can be prohibitive for sensor networks in which devices have limited capabilities.

The inability of the Internet to efficiently accommodate increased heterogeneity in the network substrates and devices has convinced many researchers to look for alternative designs such as [3, 12, 13]. These designs revisit fundamental design principles and try to provide new solutions to issues that are addressed by IP network. In particular, issues regarding inter-connectivity among multiple substrate networks are considered.

The authors of [9] argue that overlay networks provide a way to first experiment with new routing and architecture designs and then to incrementally deploy these new designs. The authors of [3] propose an implementation of this idea by introducing virtualization into the network. In this proposal, virtualization is employed to slice links and routers and other network resources in order to support multiple virtual networks simultaneously. The infrastructure provides a set of dedicated but multiplexed virtualized overlay nodes, a method first advocated by PlanetLab.

Plutarch [12] defines a framework in which the heterogeneity of the underlay networks is explicitly exposed to the upper layers and argues that there are situations in which hiding the differences between substrates under a single common overlay, similar to what is done by IP, is infeasible or undesirable. Furthermore, it is claimed that a model of networking based on the assumption of existence of heterogeneous substrates not only provides a better framework to evaluate new architectures but also captures the current state of the Internet better than the original Internet design principles [10].

2.4 Inter-Connectivity of Heterogeneous Networks

The main design objective of delay tolerant networking (DTN) [14] is to provide communication among networks with heterogeneous timing characteristics, e.g., near-earth satellite communications, sensor networks, and military ad hoc networks. Although originally, the heterogeneity in DTNs referred to the diversity in timing characteristics of networks, now the term also covers the diversity in the forwarding mechanisms, addressing schemes, and other characteristics of networks. The main concepts embodied by DTN are an overlay network that spans multiple *region*s, a globally interoperable naming system, and a novel set of transport protocols designed to tolerate a large amount of delay at each hop.

In the original architecture for DTN proposed in [14], the identifier for a node is the concatenation of its region ID, a globally known fully qualified domain name (FQDN), and its identifier in the region, the entity ID (e.g., a MAC address, or an IP address). Note that with this naming system, a node that is connected to multiple regions has a different identifier for each region. The advantage of this region-dependent address organization is that it makes it possible to implement an efficient two-level hierarchical routing. At the first level, i.e., inter-region, the routing is done solely based on the region ID, and at the second level, i.e., intra-region, the routing is done only based on the

entity ID.

Plutarch [12], which tackles the issue of heterogeneity in the Internet, proposes two abstractions: *context* and *interstitial function* (IF). A context is a region of the network that is homogeneous in some regards and contains a set of bindings with reference to which names may be resolved. The purpose of IF is to pass the data at the border of two adjoining contexts. The IF can manipulate data in order to enable an end-to-end service. The most important one of these data manipulations is the rebinding of names in the destination context. However, the IF abstraction allows any necessary manipulation at the contexts border, e.g., transcoding, inserting forward error correction, transport layer signaling, etc..

In Plutarch, the binding of names with addresses happens at the end systems. Each end system exists in an explicit context in which all usable names and addresses are bound. For example, the binding between the destination name and its address happens when the source node receives a chain of contexts as a result of resolving a destination name. If the destination is accessible through multiple contexts, multiple of these context chains are returned when resolved. Plutarch assumes that only one of these context chains will be instantiated and used at any instant.

The goal of *SpoVNet* (Spontaneous Virtual Networks) [5] is to provide a framework for building services on top of a heterogeneous set of substrate networks. This goal is achieved by providing a generic *underlay abstraction*, that is an overlay network built on top of heterogeneous substrate networks. The underlay abstraction has two layers. The lower layer, called *base communication*, is responsible for managing multiple underlay addresses that may exist for an end-point. This layer deals with mobility, and multi-homing. The *base overlay* is the second layer and provides end-to-end connectivity using a key-based routing scheme. End-to-end services such as multicast are then built on top of this underlay abstraction.

The current implementation of the base overlay in SpoVNet uses *Chord*. Chord [35],

a popular structured peer-to-peer network, organizes nodes in a ring structure and implements an efficient distributed hash table. The authors of [5] maintain that it is possible to replace Chord with any arbitrary structured overlay that supports key-based routing. Although it is possible to route all messages using the base overlay, the structured overlay may not provide the most efficient path available. Instead, the services on top of the underlay abstraction are expected to use the base overlay only to resolve node identifiers to their corresponding set of locators, and then use the communication base to establish connections between nodes.

FARA (Forwarding directive, Association, and Rendezvous Architecture) [11] is an architecture that incorporates a new organization of the concepts of naming and binding, two fundamental concepts for the current Internet and any future internetworking architecture. In FARA, communication happens between pairs of entities, which are generalization of the application concept and may span multiple physical machines. Packets are exchanged through logical links, called *associations*. FARA decouples the network layer node identity (association ID) from the underlay network locators (forwarding directive).

A main objective of FARA is to avoid introducing a new global namespace. This is achieved by assuming that the bindings between the association IDs and forwarding directives are unique only within entities involved in each particular communication. These bindings are stored locally in the entity, as part of the communication state alongside the application state. FARA envisions a rendezvous service that helps entities to discover and locate other entities for each particular service, and expects a handshake phase between two entities to establish an association and related association IDs. Furthermore, it predicts the need for mechanisms for maintaining address bindings between a node identifier and its multiple locators.

2.5 Inter-Connectivity in the Augmented IP Network

Another related body of works are motivated by the interrupted end-to-end connectivity in the Internet due to NAT and other similar middle-boxes. Among them are proposals that envision the existence of multiple address realms in the future of the Internet, e.g., TRIAD [18], IP Next Layer (IPNL) [17], and 4+4 [37]. Different from works mentioned in the previous section, which try to address a broad spectrum of diversity in the network, these works are concerned with the much narrower problem of providing connectivity among different address realms of a single network (i.e., IPv4). Although different in scope, both sets of works address a similar core issue of designing a naming and addressing systems, and related mechanisms for creation and manipulation of the name/address bindings for an environment with multiple addressing contexts.

IPNL [17] builds an overlay (or "next layer") protocol on top of multiple IPv4 realms. Enhanced NAT boxes, called *nl-routers*, forward packets between these realms. An nl-router that connects a private realm to the public IPv4 realm is known as the *frontdoor*. A triplet of the frontdoor address, realm number, and host address, called IPNL address, fully identifies a host in IPNL. However, IPNL addresses are not used as long-term identifiers, but only as short-term locators. IPNL uses fully qualified domain names (FQDN) as host identifiers. The primary reason for this approach is to not require renumbering of hosts in the private realm when the frontdoor changes. IPNL also provides a mechanism which enable one end of a connection to automatically learn about possible changes in the IPNL address of the other end. Such a mechanism supports mobility of hosts.

Since each nl-router can be connected to multiple realms, an nl-router has to deal with multiple address realms. When an IPNL address is used, the realm number identifies the correct address realm in which the host address may be used to locate the destination.

Since each realm is associated with its own globally unique domain name, the FQDN also uniquely identifies the address realm. In addition, each end system can be connected to more than one realm at the same time, but this results in multiple FQDN for that host, one for each realm. Note the similarity of this design to the two-level hierarchical addressing schema in a DTN. The difference is that, in DTN, the host address can be of any type while, in IPNL, it is either a host name or a private IP address. Further, in DTN, the domain portion of the address is always an FQDN while, in IPNL, it can be either an FQDN, or a tuple of the frontdoor address and realm number.

Similar to IPNL, 4+4 [37] also deals with the problem of interconnecting IPv4 address realms. However, unlike IPNL, each front door is connected to only a single realm, which makes realm identifiers unnecessary in 4+4. It uses a pair of IP addresses, which consists of the frontdoor public IP address and the host private IP address, as the host identifier. 4+4 only uses DNS to learn the two addresses of each host, and therefore, FQDNs are not host identifiers as they are in IPNL. 4+4 packets are minimally encapsulated IP packets. They contain two 64-bit fields for the source and destination addresses. The fields are placed such that the outer header always contains addresses that are understood by the IPv4 routers in the realm the packet is transiting; that is, in a private realm the private half of the extended address is visible in the outer header, while in the public Internet, the public half is visible. This ensures that routers can forward packets toward the destination without understanding 4+4. Thus 4+4 requires changes only in NAT boxes, and no changes at end-systems or routers.

TRIAD [18] is another internetworking architecture that addresses the lack of end-to-end connectivity caused by NAT and other middle-boxes. It proposes a content layer which uses FQDNs as node identifiers. TRIAD uses source-based routing to route messages based on the destination domain name. Forwarding tables in relays are indexed with domain names instead of IP addresses. The *Name-based Routing Protocol* (NBRP) is a routing protocol which is responsible for distributing routes to domain names among

relay nodes. The *Internet Relay Protocol* (INRP) is a protocol that uses the routing information maintained by relay nodes to resolve a name to a list of relay addresses. Both INRP and NBRP can be implemented as extensions to the current domain name system.

To establish communication between two nodes, the sender node uses INRP to send the first message to destination domain. This is done by having each relay forward the message to the next-hop relay node based on routing tables which are indexed by domain names. The last hop, i.e., the authoritative relay of the destination domain, will deliver the message to the destination node. The reply message which travels the request path back to the sender contains addresses of all relay nodes. The sender uses this chain of addresses for subsequent packets. As two NAT-extended architectures, TRIAD and IPNL are similar in many aspects of their naming and addressing. Unlike IPNL, TRIAD combines the naming and routing infrastructures and can support more than two levels of hierarchy among domains.

2.6 Summary

We discussed the related work on inter-connectivity of heterogeneous networks, which are motivated by networks with multiple address realms and namespaces. Although some of these works have different foci, common to them is the design of flexible naming and addressing systems. The naming system supports multiple addressing realms at a particular layer with a dynamic binding that allows mobility and multi-homing. Examples of such addressing realms are *context* in Plutarch, *connectivity domain* in SpoVNet, *region* in DTNs, and *realm* in IPNL.

With the existence of multiple addressing realms, bridging across these realms becomes a critical function of the architecture. There are two fundamental alternatives to bridge between addressing realms: *translation* and *common namespace*. With transla-

tion, gateways between networks translate a unique identifier from one address realm to a unique identifier in another. NAT is a common example of bridging by translation. In the common namespace approach, the identifier used at a particular level belongs to a common namespace shared by all networks. The local locator are then mapped to these identifiers which eliminates the need for translation. This is the approach taken by the original IP network, and favored by almost all inter-networking proposals. The difference between these proposals lies in the scope of this namespace and the mechanism by which the common identity is translated to the local locator [4].

For example in DTN, the common namespace is global (FQDN based) and with a late binding between the name and the address at each realm boundary. IPNL and 4+4 operate in a similar way, except that there is a certain structure among different address realms which makes it possible to design a more efficient binding process. In FARA, the common namespace is shared only among communicating entities, and has a small scope. In SpoVNet, the common namespace is shared among all nodes in the base overlay, the common identifier is resolved to another globally unique locator by the source, which is then used for routing the message to the destination address realm. The abstraction defined by Plutarch, allows either one of these two alternatives to be implemented. The IFs at boundaries of address realms can either translate one node identifier to another, or simply rebind the same identifier to another network locator in the new context.

In our work, we do not define a common namespace but rather allow the overlay protocol to define its own namespace. Similarly, we do not try to propose a specific solution for inter-connecting multiple address realms (i.e., substrate networks) and it is left to be a responsibility of the overlay topology. Instead, we are interested in identifying fundamental functions, primitives, and mechanisms that can be used by various solutions. In particular, we look at methods for efficient propagation of address bindings in the overlay network.

Chapter 3

Overlay Networks in a Multi-Substrate Environment

In this chapter we discuss how the fundamental concepts of overlay networking have to be modified to suite overlay networks that operate over multiple heterogeneous substrate networks.

Since our design and implementation is part of the HyperCast project we take concrete examples from this project. The HyperCast project defines a set of abstractions and primitives for overlay networking. HyperCast provides a comprehensive implementation based on this abstractions that allows rapid development of new overlay protocols [25].

3.1 Overlay Networking Concepts

An overlay network is essentially a network built on top of one or more underlay networks (also referred to as *substrate network*). Nodes in the overlay network are connected to each other through logical links. Each logical link corresponds to a path, which may consist of many physical links, in an underlay network. Application layer peer-to-peer networks are examples of overlay networks which are built on top of the transport layer of the Internet.

In HyperCast, any packet based communication channel can be a substrate network. This channel should provide either a unicast or multicast packet delivery service. It should also be able to provide an address for the sender of any incoming message. Form and function of this address are discussed later in this chapter. HyperCast does not assume the availability of bi-directional communication[2] in the substrate network.

This minimal set of requirements allows HyperCast to use a broad range of substrate networks. For example TCP/IP, UDP/IP, IP, Ethernet, 802.11 networks, and even another overlay network can all be used as underlay network in HyperCast. This makes HyperCast suitable as a potential platform for building overlay networks with multiple heterogeneous substrates.

3.2 HyperCast

As discussed, HyperCast provides basic components that help distributed applications to construct overlay networks based on different overlay protocols over a single substrate network. HyperCast provides an application programming interface (API) similar to that of the widely used BSD sockets, which is called the *Overlay Socket*. And overlay socket allows the application to create a new overlay network, join and leave an existing overlay network, and send data to and receive data from other members of the overlay network.

HyperCast hides from the application the complexity that is involved in maintaining the overlay topology and forwarding the application data messages. This transparency reduces the complexity of API for overlay sockets.

[2]In a bi-directional communication, a node which receives a message should be able to send a message back to the sender

3.2.1 Overlay Socket

An overlay socket in HyperCast is an end-point of communication in an overlay network. The application can use overlay socket to send application data messages to other members over the overlay network. An application starts using an overlay network by creating an overlay socket. To organize the overlay network topology and for various other tasks, overlay sockets exchange control messages (protocol messages) among each other.

The design of the overlay socket is modular. HyperCast isolates various functions in overlay networks and defines interfaces for them. This separation of functions (e.g., message forwarding, security, topology maintenance and etc) enables the modular design in which an implementation of a common function can be used with various overlay protocols. This minimizes the effort required to design and implement a new overlay network. The main components of the overlay socket include the overlay node, the forwarding engine, the adapter, the interface, and the address repository. Figure 3.1 shows the main components of an overlay socket and their relationship to each other.

3.2.2 Overlay Node

The overlay node in HyperCast handles discovery of other nodes, establishing neighborhood relationships, and maintaining logical links between neighbors. The overlay node is the only component of the overlay socket that is aware of the overlay topology.

Overlay nodes exchange protocol messages with each other to construct and maintain the logical topology of the overlay network. The format and content of these messages are determined based on the type of the overlay protocol that the node implements. Although the internal functionality of the overlay node depends on the protocol it implements, all types of overlay nodes present the same interface to other components. HyperCast requires the overlay node to be able to make forwarding decisions and calculate the next hop node for each incoming message using only the destination of that message and its

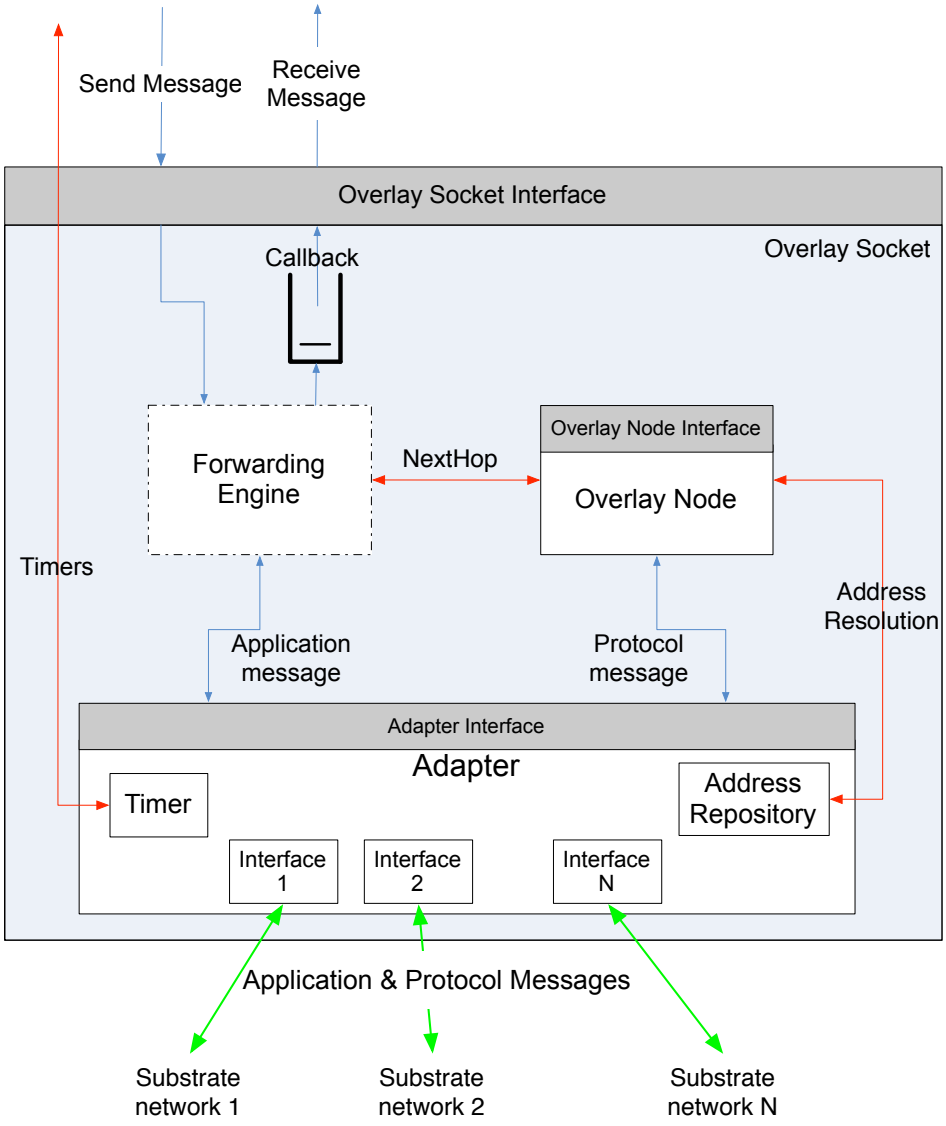


Figure 3.1: An overlay socket and its components.

local information about the overlay topology.

3.2.3 Adapter and Interfaces

Each overlay socket has one adapter, which provides an abstraction layer of the substrate networks. Every other component of the overlay socket uses the adapter to send and receive messages to destinations identified by *substrate addresses*. In addition to providing

send and receive functions, the adapter is responsible for the creation and manipulation of substrate addresses. These addresses are treated as opaque objects by other components of the overlay socket, e.g., the forwarding engine and the overlay node. Section 3.2.4 contains more information on substrate addresses.

The adapter maintains two buffers for incoming messages, one for protocol messages and one for application messages. Incoming and outgoing messages are not processed by the adapter. The adapter treats them as byte arrays. The adapter contains a timer thread which provides timing service for all components of the overlay socket.

Internally, the adapter has one interface for each substrate to which the overlay socket is connected. Each interface implements the send and receive functions specific to that substrate and is able to construct and manipulate the substrate specific address format. Examples of types of interfaces are UDP Unicast interface, UDP Multicast interface, TCP interface, and Ethernet interface. There is a different interface type for different communication protocols, e.g., TCP and UDP. Moreover, if a protocol supports multiple delivery modes (e.g., multicast and unicast) there is a separate interface for each delivery service. Finally, although discouraged, an adapter may contain multiple interfaces to the same substrate.

3.2.4 Addressing Structure

HyperCast defines two types of identifiers. The *logical address* uniquely[3] identifies an overlay node within a single overlay. The *underlay address* identifies a network attachment point (or multiple ones in case of a multicast or anycast address) in an underlay network and is used to send messages in that underlay substrate network. In Hypercast, the overlay node identifier, i.e., the logical address, is decoupled from address(es) of its network attachment point(s), i.e., the underlay addresses.

[3]In some cases, it is possible for a logical address to lose its uniqueness temporarily for a short period of time. Such incidents are considered an address conflict and are resolved by the overlay protocol as soon as detected.

Applications that use an overlay network and most internal functions of the overlay network such as naming, routing, and topology maintenance use logical addresses. The underlay addresses are kept and maintained only in the adapter in the overlay socket. These addresses are treated as opaque objects by all other components of the overlay socket and are presented to the adapter in order to send a message. The adapter provides an API for creation and manipulation of underlay addresses.

Logical Address

A logical address uniquely identifies an overlay node in an overlay network. In HyperCast, each node is expected to be able to provide a next hop destination for each message using only the destination's logical address of that message.

Underlay Address

An underlay address is an identifier that locates an end-point or multiple end-points in a substrate network. The forwarding mechanism of the substrate network uses an underlay address in order to deliver a message. HyperCast supports a range of options for the function and form of an underlay address. For example, an underlay address may be location dependent. It is possible for the forwarding mechanism to modify the source or destination underlay address on the fly. In the following, we list general characteristics of the underlay address in HyperCast.

- An underlay address must be expressed in a well-defined syntax.
- An underlay address must be *reversible*, meaning that the source underlay address of an incoming message can be used to send a reply back to the sender of the message. For example a public IP address is a reversible address.
- An underlay address may not uniquely identify a single destination. In particular a multicast group address is an underlay address.

- An underlay address may be composed of several sub-addresses, each of which is a locator in a smaller scope, e.g., an <IP, Port number> is an address that is composed of two locators: an *IP address* which locates a host in an IP network, and a *port number* which locates a service on a host.

- An underlay address may be dependent on the location of the entity it identifies in the substrate network. For example the IP address of a host in the Internet depends on its location. Such addresses often make it possible to have an efficient forwarding.

- An underlay address does not guarantee global reachability in its substrate and may describe a forwarding process that fails to deliver the packet from some parts of the network. For example a MAC address in a wireless 802.11 ad hoc network may be unreachable from some parts of the network when there is a partition.

HyperCast requires substrate networks to be able to provide the source underlay address of any incoming message. This requirement along with the reversibility of the address guarantees that the receiver of a packet is able to contact its sender through the same substrate.

Address Bindings

Each overlay node has a separate underlay address for each attachment point to a substrate network. The association between an overlay node identifier (i.e., logical address) and the identities of its network attachment points (i.e., underlay addresses) is expressed in the form of a binding. Each node in the overlay network maintains a set of such address bindings for all other nodes with which it communicates.

Overlay protocols designed for a single substrate environment simply assume that each overlay node has only one network attachment point. This assumption greatly simplifies the design of mechanisms that are responsible for maintaining address bindings. However

in an multi-substrate environment, each overlay node can potentially be attached to several network attachment points, each belonging to a different substrate. This requires address bindings which are more complex, and thus, more sophisticated mechanisms for obtaining and maintaining them.

An alternative approach is to allow each overlay node to have multiple logical addresses, one per each substrate. This is the approach taken by some delay tolerant networking architectures [14], where each node has an identifier for each domain to which it is connected. The advantage of this approach is that the overlay protocol can use the same one-to-one address bindings and relatively simple mechanisms to maintain them.

This approach essentially delegates the problem of having multiple identifiers for a single service to the distributed application which runs on top of the overlay network. In this approach, the distributed application has to deal with complexity of maintaining the association between the identifier of each of its instances with multiple identifiers of its underlying overlay socket.

Another reason in favor of solving the problem of multiple underlay identifiers at the overlay layer instead of higher layers (i.e., distributed application) is that the overlay is operating directly above the substrates. This direct access enables the overlay network to make intelligent decisions when resolving a logical address to an underlay address by choosing the underlay address from the substrate that provides the best service for the application.

In the light of the above discussion, having the overlay network deal with one-to-many address bindings and other related issues that arise in a multi-substrate environment can reduce the complexity of the design of the distributed applications. In the next section, we discuss the implications of having multiple substrates on the address scheme. The detail of our design is discussed in Chapter 4.

Substrate Independent Address Format

As discussed in Section 3.2.4, each underlay address describes a forwarding process that is useful only in the context of its substrate. As we will discuss later in Section 4.1.1, it is often necessary to carry underlay addresses in an overlay protocol message to enable overlay nodes to self-organize into a network topology.

In a multi-substrate environment, it is possible for a message that is transmitted in a substrate network to contain underlay addresses which do not belong to the carrier substrate. As such, there is no guarantee that a node forwarding this message or its receiver knows how to interpret and handle the underlay addresses included in that message. Therefore, it is necessary to have a uniform and well defined way of representing addresses across all substrates. We refer to such a representation of an underlay address as a *Substrate Address*. An overlay socket parses and interprets a substrate address consistently, independent of the substrate to which it belongs. This allows messages to contain substrate addresses with an unknown format without risking incorrect parsing or interpretation of these addresses.

A substrate address contains information that identifies its substrate in addition to the underlay address which locates the end point in that substrate. The identifier for the substrate consist of an *address realm* and a *protocol type*. The design and implementation of substrate address is discussed in Section 4.1.3.

3.2.5 Address Repository

The address repository is the component in the overlay socket that maintains address bindings. The address repository is a sub-component of the adapter and can only be accessed through the adapter. The address repository provides a resolution function that resolves a logical address to a substrate address. Additionally, it allows other components (e.g., the overlay node) to add new address bindings, and remove or update existing

bindings.

In the previous version of HyperCast (Hypercast 3.0), address bindings are kept by the overlay node and can only be modified by the overlay node. By introducing the address repository, it is now possible for other components other than the overlay node to access and modify address bindings. The separation of address binding maintenance from the overlay node makes it possible to design a component that deals with these bindings and that can be used with multiple different protocols. Lastly, it simplifies the design of new overlay protocols.

3.2.6 Forwarding

In Hypercast, forwarding decisions are made by the overlay node exclusively based on the destination logical address for unicast messages and the source logical address for multicast messages. The actual forwarding mechanics are implemented by the forwarding engine. By separating the mechanics of forwarding from the routing process, HyperCast separates the control plane of the overlay network from its data plane.

When an overlay socket receives an application message it passes the message to the forwarding engine. If the message is destined to the local overlay socket it is passed to the application. Additionally, if the message has to be forwarded, the forwarding engine requests the node to provide the next-hop or next-hops and forwards the message.

3.3 Self-Organization in a Multi-Substrate Environment

A key strength of overlay networks is their ability to self-organize. In general, self-organization in networking means that the network must be able to spontaneously organize toward desirable global properties without a central entity with global information.

Scalability, low maintenance, and robustness of self-organizing solutions makes them desirable for distributed applications.

One area of overlay networking in which self-organization is used, is in the creation and maintenance of the topology of the overlay network. In an overlay network, the topology dictates the neighborhood relationships of each node. We distinguish two separate functions involved in the creation and maintenance of the topology in self-organizing overlay networks.

The first function provides necessary mechanisms for new nodes to join the overlay network, and is referred to as *bootstrap* or *rendezvous*. During rendezvous, a new node either contacts an existing member node or requests that a member node contacts it, in order to join the overlay network. In HyperCast, two different methods are used by overlay protocols to bootstrap. A node either contacts a known entity, a peer (i.e., buddy) or a server, or broadcasts a message. The first method requires the node to have a pre-configured address for at least one peer or server. The second method requires a multicast substrate to be available.

The second function is responsible for maintenance and improvement of the established topology in order to preserve its desired structure and is referred to as *topology maintenance*. For example, a structured peer-to-peer overlay protocol such as Chord tries to organize nodes into a logical ring and maintain that structure. This is done by nodes periodically communicating with their neighboring nodes. It is through these communications that nodes learn about changes in the network topology due to node departures, node arrivals, partitions and other events. Nodes cooperate and make necessary modifications to their neighborhood to ensure that the overlay topology is not affected. It is important to realize that the bootstrap and topology maintenance functions are protocol dependent.

In both these functions, nodes rely on various methods to obtain substrate address information of other nodes in order to potentially establish neighborhood relations with

them. One method for obtaining substrate address information is to use an out-of-band mechanism, for example through configuration, via a server, or a discovery service such as DHCP. The drawback of relying on out-of-band mechanisms is their reliance on manual configuration and external infrastructure. Another method is to use broadcast communication. In this method, each node broadcasts messages which include its own address and some topology-relevant information and waits to receive broadcast messages from other nodes. Bootstrap function almost exclusively uses the two previous methods because they do not require existence of any overlay links.

Finally as the third method, nodes use already established links in the overlay network to exchange substrate address information. The topology maintenance function uses this method to exchange substrate address information. This method uses a mechanism which we refer to as a *node advertisement*. A node advertises the address information of itself or that of another node by including address information in a protocol message sent to other nodes. This substrate address information allows the receiver of the node advertisement to be able to contact the advertised node. Section 4.1.1 contains an elaborate definition for the node advertisement in the context of HyperCast and includes some examples of it.

As discussed above, mechanisms for self-organization of a topology in an overlay network depend on nodes obtaining substrate addresses of other nodes, or availability of broadcast communication. For this reason, self-organization mechanisms that are designed for a particular substrate and are effective on that substrate may not be effective in a multi-substrate environment. Therefore, an overlay network needs additional mechanisms to be able to self-organize and operate in a multi-substrate environment.

Chapter 4

Cross-Substrate Advertisement

Cross-substrate advertisement (CSA) is a set of mechanisms by which overlay sockets exchange address information of one substrate network across another substrate network. CSA is needed in a multi-substrate environment to assist overlay nodes in learning about alternative substrate addresses by which other overlay nodes can be reached.

As discussed in Chapter 3, an overlay node obtains one or many substrate addresses of another overlay node from two sources: from members of the overlay network via exchange of protocol messages, or from out-of-band sources that are not part of the overlay network (e.g., configuration, or a server). Regardless of the source of the substrate addresses, they are useful to the receiver only if the receiver is connected to their respective substrates.

In a multi-substrate overlay network, it is important for nodes to receive the substrate addresses that are useful, otherwise it negatively affects nodes ability to construct and maintain the topology. A simple and inefficient approach is to always send the complete list of substrate addresses of a node anytime its address information has to be included in the message.

CSA is a set of mechanisms that assist overlay nodes in obtaining substrate addresses of other nodes. The mechanisms supplement existing methods that overlay protocols use

to exchange substrate address information. They determine when, and which substrate addresses are exchanged.

In the following, we present two examples to illustrate the potential benefit of cross-substrate advertisement in an overlay network with multiple substrate networks.

Example 1.

In the situation depicted in Figure 4.1, two nodes, A and B, are directly connected through two substrate networks. The first substrate (S_1) is capable of broadcast communication, but it does not provide fast unicast communication. The second substrate (S_2) provides fast unicast communication, but does not support broadcast. B prefer S_2 for exchanging protocol and application messages but it does not have address of A on S_2. B uses S_1 to broadcast a rendezvous message in order to discover A. This message which is carried on substrate S_1 contains substrate address of B on S_2. Once A receives this message, A can contact B over S_2 in order to establish a neighborhood. Note that in this case, it is sufficient for A and B to exchange only their own address information across the substrates.

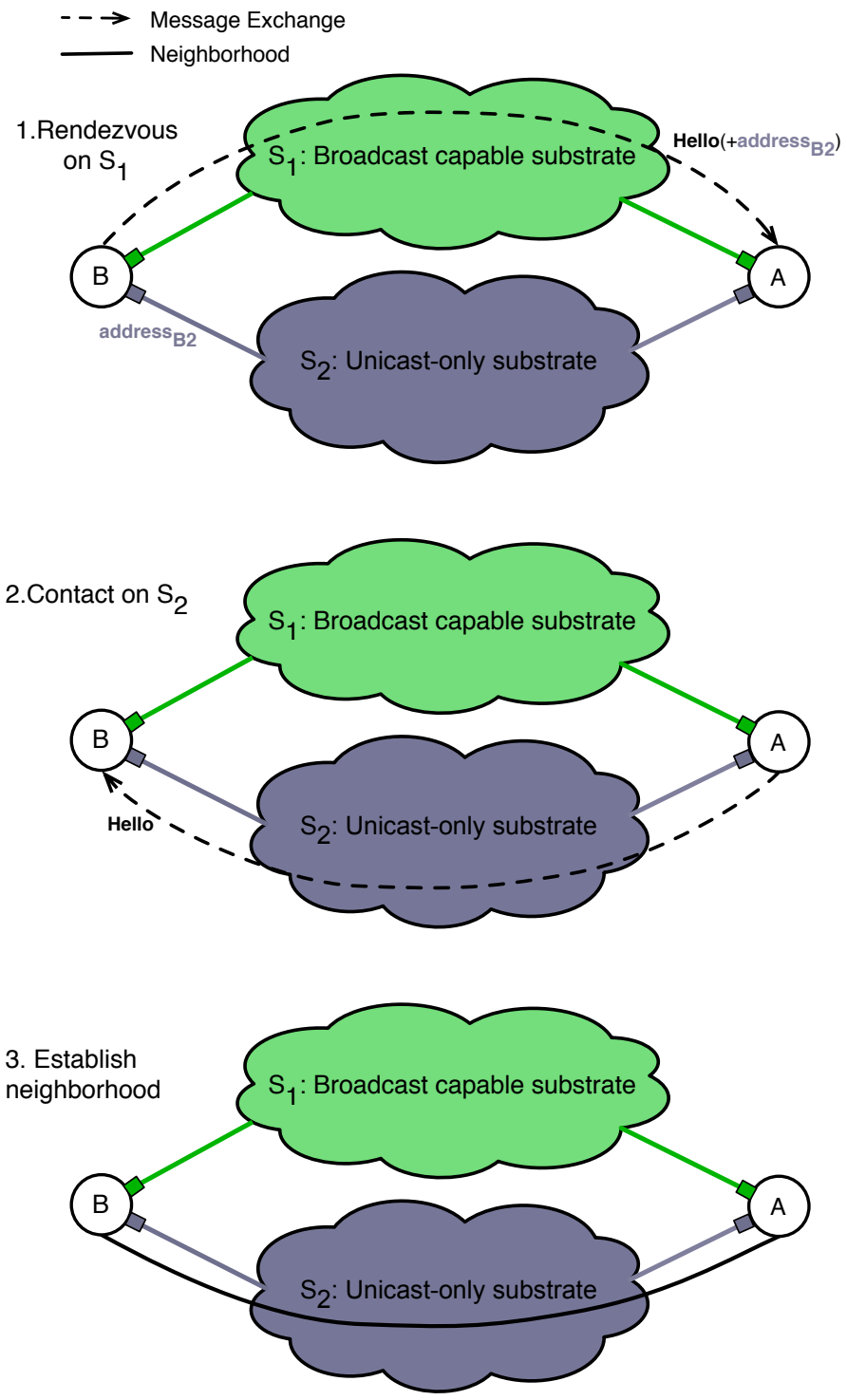


Figure 4.1: CSA can improve the rendezvous process for two nodes that are directly connected via multiple substrate networks.

Example 2.

In Figure 4.2 three nodes and three substrate networks are illustrated. Node A has a neighborhood relation established with node B over substrate S_2. Node C has joined the overlay network over substrate S_3 and with B as neighbor. However, node C prefers substrate S_3, and as it can be seen in the example it is possible for nodes A and C to establish a neighborhood relation over substrate S_1. Suppose C learns about A through a protocol message by B. The CSA ensures that this message includes address information of A for substrate S_1. After receiving the message, node C can contact A on substrate S_1, and it establishes a neighborhood relation with node A over substrate S_1. This scenario is inherently more difficult than that of Example 1, since it is not sufficient that A and B, or B and C to exchange their own address information with each other. Here, the address information of a third party node (i.e, A) exchanged between B and C across substrate networks. Also note that node B which advertises the substrate address information of A for S_1 does not itself have any connection to the substrate S_1.

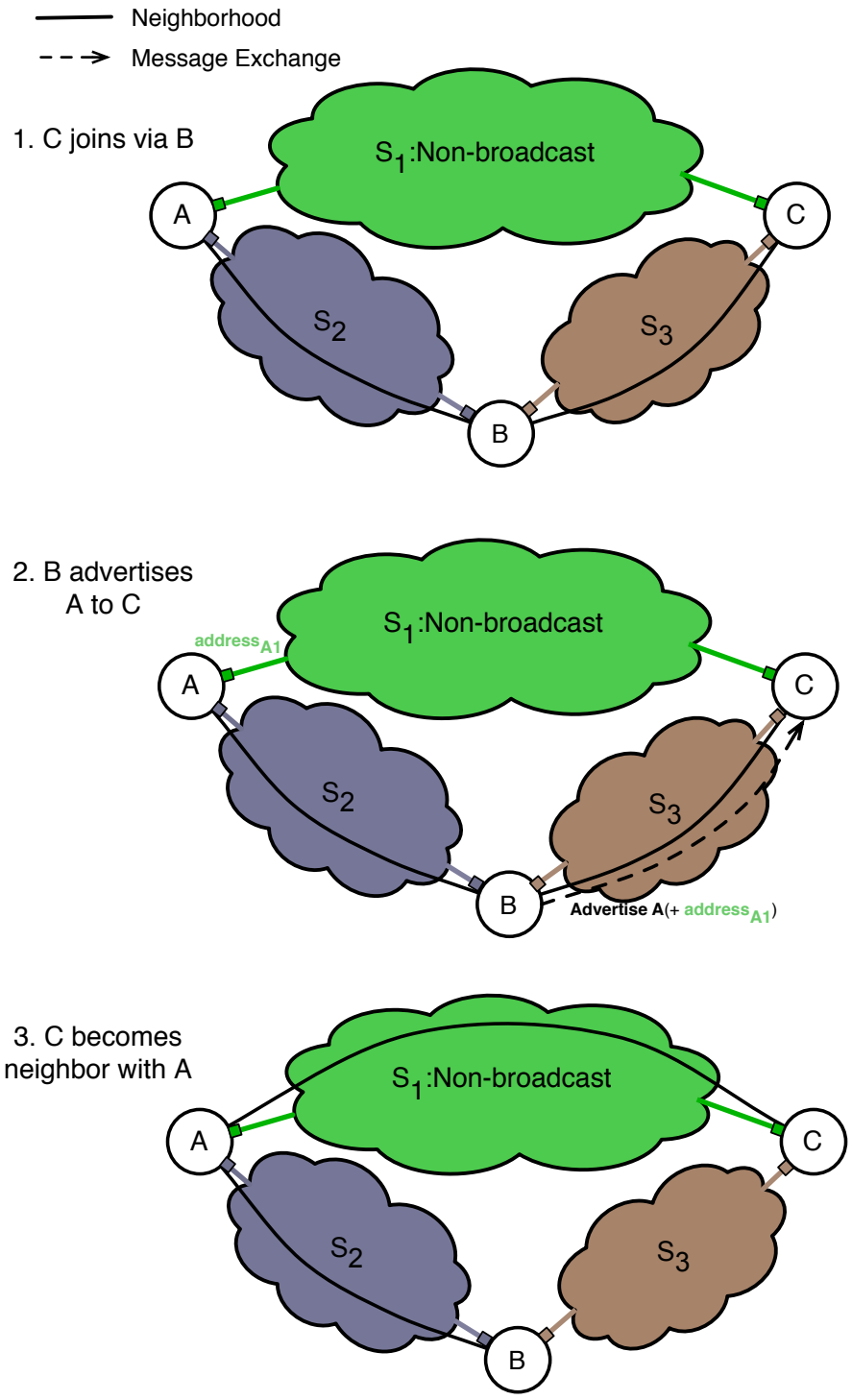


Figure 4.2: CSA improves connectivity and self-organization of nodes in different substrate networks.

As demonstrated by these examples, exchanging address information across substrates is crucial for self-organization of overlay networks in a multi-substrate environment. Mechanisms for cross-substrate address exchange can be either part of the overlay protocol itself or operate independently. We propose a set of protocol independent mechanisms that efficiently disseminate substrate addresses of overlay nodes. The primary benefit of our approach is that it can be used with multiple overlay protocols without requiring any modification to their rendezvous and self-organization methods.

4.1 Design

Cross-substrate advertisement has two main objectives. The first objective is to provide necessary means which allow a node to obtain the complete substrate address information of another node with whom a connection is already established. This is achieved by *Direct Address Exchange* methods, in which two nodes that communicate directly exchange their substrate address information with each other. The direct address exchange ensures that a node is able to obtain the complete list of substrate addresses of its neighbors and therefore becomes aware of alternative paths that exist between them.

The second objective is to provide necessary means for a node to be able to obtain the complete substrate address information of an advertised node. This is achieved by *Third-party Address Exchange* methods, which ensure that when a node receives a node advertisement it also eventually receives the complete list of substrate addresses of that node. This way the receiver knows about all substrates through which the advertised node can be reached, and it is able to contact the advertised node if they share at least one substrate.

An important aspect of our design is that we require address exchange methods to be protocol independent. The main advantage of this approach is that the same methods can be used by multiple overlay protocols. Another advantage of this design is that

it simplifies the design of the overlay protocols. Finally, this design makes it easier to adapt overlay protocols designed with a single substrate network to a multi-substrate environment. The drawback of this approach is that it may not be as efficient as a design which takes into account the specific behaviors of an overlay protocol.

4.1.1 Node Advertisement

A node advertisement is a protocol message that contains one or more substrate addresses of one or multiple nodes. Nodes send advertisement messages with the intention of enabling the receiver node to contact the advertised node using the included substrate address information.

As discussed in Section 3.3, node advertisements are used by self-organizing overlay protocols as a method for exchanging substrate address information that are used for rendezvous and topology maintenance functions. Often nodes include other information about the advertised node in addition to substrate address information. This additional information provides context that helps the receiver node decide when and how it will use the advertised substrate address information. For example, in HyperCast protocols, an advertisement contains send the logical address of the node along with the substrate address information.

In the following, we define some terms that are associated with node advertisement.

Advertisement message

The protocol message in which at least one node is advertised. The advertisement consists of a logical address and one or more substrate addresses.

Advertised node

A node whose substrate address information is being advertised.

Advertisement pair

Advertisement pair refers to the pair that consists of the logical address and substrate address information of an advertised node.

Here are few concrete examples that showcase node advertisement in various overlay protocols.

Delaunay Triangulation (DT) protocol: Each DT node, in each heartbeat, advertises to each one of its neighbors, their respective clockwise (CW) and counter-clockwise (CCW) neighbors in a `HelloNeighbor` message. These advertisements include the logical address of the nodes along with their substrate address information. The receiver node contacts both advertised nodes to form a neighborhood relation with them.

Clustering (CT) protocol: In the CT protocol, a message known as the `ReferralResponse` message is sent by a cluster head to a cluster member or another cluster head in order to inform them about other cluster heads. The message contains entries that have information about other heads that the sender knows about. Each entry is a node advertisement as it contains the substrate address information of a head. Additionally each entry includes the logical address of the head, and additional information about the service it provides to its member nodes. In this case, the node advertisement is the primary function of the `ReferralResponse` message.

Pastry protocol: In Pastry, nodes sometimes exchange their *leaf set*. The leaf set of each node is a set of close nodes. For example when a new node joins the network, it asks an existing member node for its leaf set and uses this information to populate its own leaf set. A similar exchange happens when a node departs. In addition to other information, each entry in the leaf set contains both the 128-bit node identier which indicates the node's position in a circular space (i.e., logical address) and substrate address information of a Pastry node. Leaf set entries are considered node advertisements with our definition.

4.1.2 Substrate Address

A substrate address consists of an underlay address and information that uniquely identifies the substrate. It is possible for an underlay address to be mapped to different nodes in different address domains. For example an private IP address can be used for different host in different private networks. Thus, it is necessary to include information that identifies the address domain and the scope within which the address uniquely identifies an end-point. We refer to this address domain and scope as the *address realm*. Examples of address realms are the public IPv4 network, any private IPv4 networks, any IPv6 network, etc. In particular, in cases where there are multiple instances of the same network, e.g., private IP networks, the address realm should include additional identification.

It is also necessary to identify the protocol type. Together, the underlay address, the address realm identifier, and the protocol type can uniquely identify a communication end-point in an environment with heterogeneous substrates and communication protocols. We use the following format for substrate addresses:

Substrate Address:

<address realm identifier, protocol type, underlay address>

Binary Form:

[hash(address realm, protocol type), underlay address length, underlay address]

Since the address realm identifiers and protocol types can have variable length, we use fixed sized hash values when transmitting them in the messages. While the length of the underlay address can be deduced from its substrate type, this is not done in practice as it requires the length of underlay address for all possible address realms or protocol types to be known globally.

In HyperCast, each interface in the adapter should be able to provide the overlay socket with the tuple of <address realm, protocol type, underlay address>. Table 4.1 contains examples of such tuples for various interface types.

Table 4.1: Substrate address examples.

Host Address	Interface Information	Substrate Address
128.100.100.128	TCP, port: 8080	<public_ip, tcp, 128.100.100.128:8080 >
192.168.0.10	UDP Unicast, port: 8001, private network: mylan	<private_ip_mylan, udp, 192.168.0.10:8001 >
192.168.0.10	UDP Unicast, port: 8001, private network: ut12	<private_ip_ut12, udp, 192.168.0.10:8001 >
10.0.0.10	UDP Multicast, multicast group: 224.228.19.78, port: 9999	<private1,multicast_udp, 224.228.19.78:9999>
FF:FA:34:09:41	Ethernet, multiplexing number: 1234	<EUI-48,multiplexed_ethernet, FF:FA:34:09:41/1234 >
(100,100)	Overlay Interface, overlay ID: test_overlay, protocol: DT	<test_overlay, DT, (100,100) >

The substrate address is the only form of address that is exchanged by the CSA methods. The binary representation of the substrate address can be carried in the payload of any type of message and via any substrate network. In a multi-substrate environment, an overlay node may not be able to use all of the substrate addresses which it has received, but only those for which it has an interface with matching address realm and protocol type.

4.1.3 Address List

In a multi-substrate environment, each overlay node is potentially connected to multiple substrates and therefore has a list of substrate addresses. The *address list* of a node is a list of substrate addresses that belong to that overlay node. The address list does not necessarily contain all substrate addresses of an overlay node. The following is the format of address list:

> **Address List**: <Number of Addresses, <[Substrate Address #1, Preferences], ..., [Substrate Address #N, Preferences]>>

Each substrate address in the address list comes with two preference values (each 4 bits long). These values are set when the address list is created for the first time by its owner and are the preferences of the owner for receiving data and protocol messages on that substrate address. A higher value represents a better preference. These values are used by the address repository when it resolves a logical address to substrate addresses.

An overlay node does not need to have the complete address list of another overlay node to be able to contact it. For example an overlay socket that is connected to only one substrate can only use substrate addresses the belong to that particular substrate, and thus only needs to receive those addresses. However, there are cases in which addresses not used by an overlay node have to be kept by that node. In particular, this is the case for node advertisements. When a node wants to advertise another node to one of its neighbors, it may have to send substrate addresses not used by itself, to allow the receiver to be able to reach the advertised node.

4.2 Address Exchange Methods

The CSA mechanism consists of two categories of address exchange methods: direct address exchange methods and third-party address exchange methods. Direct address

exchange methods enable a node to obtain the address list of its neighbors and nodes with whom it communicates. These exchanges ensure that the node can always choose the best available substrate to contacts its neighbors. The third-party address exchange methods are used when addresses are exchanged in a node advertisement.

4.2.1 Direct Address Exchange

The efficiency of direct address exchange methods in dissemination of address lists depends heavily on the type of the substrate network, in particular whether it is possible to send broadcast messages. In order to be able to adapt to a variety of situation, we have developed different strategies for direct address exchange.

Request Method

Whenever a node receives a protocol message from another node for the first time it triggers a process that sends a CSA request message to obtain the sender's address list. After receiving a request, the sender responds with its address list. The validity of the address list is limited and set by a configuration parameter. If a node receives a message from another node whose address list lifetime is expired, it sends another request for the address list. Since the request is triggered by an incoming protocol message, the pattern of requests adapts to the pattern of protocol messages. In particular, when there is no communication between two nodes, there will be no address list exchange. An issue with this method is that it is not suitable for broadcast messages, since a single request may trigger the transmission of many address lists to the requesting node which may overwhelm the requesting node with address lists.

Broadcast Method

In this method, each node broadcasts its address list as a CSA message. This is done either periodically or every time a broadcast message is sent. In the latter case, the CSA

message that contains the address list is piggy-backed on the protocol message which is broadcasted. In the former case, the CSA message is sent as a separate message. This method does not take into consideration whether the receiver nodes already have the sender's address list. The primary issue with this method is that the volume of broadcast traffic can become large.

Request/Broadcast Hybrid

If an overlay socket is connected to both unicast and multicast substrates, a combination of both request and broadcast methods can be used.

4.2.2 Third-party Address Exchange

The third-party address exchange ensures that when a node receives a message that contains a node advertisement, it also receives the address list of the advertised node. There are three different methods for the third-party address exchange: *push*, *pull*, and *gossiping*. In the push method the address list of the advertised node is included in the protocol message that contains the node advertisement. In the other two methods the address list is sent in a separate message.

Push Method

In this method, the complete address list of the advertised node is paired with its logical address and is carried in the protocol message that contains the node advertisement which means that no CSA message is required to exchange the address list. This also means that no state is kept either by the sender, receiver, or third-party nodes. This method is the simplest way to exchange the address list of advertised nodes.

The main drawback of this method is that the complete address list of the advertised node is transmitted for all node advertisements even when the receiver does not need it, e.g., it has received the address list before, or it is not going to use the node advertisement

at all. More details on the implementation of this method are available in Section 5.2.3.

***Push-Single Variation*:** This method is a variation on the push method. In this method, only one substrate address is sent in node advertisements. The substrate address sent is the most preferred substrate address according to the preference values that come with substrate addresses in the address list.

Pull Method

In this method, the address list of the advertised node is exchanged upon request by the receiver of the node advertisement. In pull method, no substrate address is sent in the advertisement message when the logical address of a node is advertised. Instead, whenever the receiver of the node advertisement wants to contact the advertised node, the address list of the advertised node will be requested from its advertiser.

The advantages of this method is that the address list is sent only when it is needed. Also the address list will not be requested again until it becomes stale even if it is advertised multiple times by different advertisers.

One drawback of this method is that two CSA messages are needed to obtain the address list of each advertised node. This makes the pull method unsuitable for cases in which a large number of nodes are being advertised by an overlay node (e.g., head referrals in CT protocol). More details on the implementation of this method can be found in Section 5.2.4.

Gossiping Method

The gossiping method is not triggered by node advertisements, but instead it is running as a background process. The gossiping method is similar to the pull method in not pairing any substrate address with logical addresses in node advertisements. However, it relies on a different mechanism to obtain the address list.

With gossiping, each node periodically sends a randomly chosen set of address lists

to a randomly chosen destination. This process gradually populates each node address repository. Eventually, each node will have the address list of all nodes that have been advertised to it and can successfully contact any of them. The main advantage of this approach over the pull method is its simplicity. Its advantage over the push method is that it generates less traffic at the cost of increased delay.

The main disadvantage of this method is that there is no guarantee that the address list of an advertised node is available when it is needed by a node. The other disadvantage is that in large networks, the gossiping process may require nodes to keep a large number of address lists. This can be mitigated by limiting the size of the repository that stores these gossiped address lists and removing unused address lists from the address repository. More details on the implementation of this method can be found in Section 5.2.5.

Push-Single/Gossiping Hybrid Method

It is possible to combine the gossiping method with the push method. Here, the push method is used to exchange only one substrate address rather than the complete address list. The nodes rely on the gossiping process in the background to provide them with the complete address lists. Note that dissemination of complete address lists by the push method renders the gossiping method unnecessary.

In this method, the receiver of a node advertisement initially attempts to use the substrate address obtained through the push method to contact the advertised node. If this attempt is successful the node can obtain the complete address information by using a direct address exchange method. If this attempt fails, the receiver node has to wait until it receives the address list through the gossiping process.

Chapter 5

Implementation

In this chapter, we discuss the implementation of a protocol for cross-substrate advertisement for the HyperCast overlay software system. There are two design objectives that have influenced the CSA implementation. First, CSA is designed to be protocol independent in the sense that it can be used by any protocol that establishes an overlay topology. Second, the implementation of the CSA requires minimal modifications to other HyperCast components.

5.1 CSA Processor

The challenge of realizing cross-substrate advertisement mechanism without expensive modifications to overlay topology is that CSA requires access to all outgoing and incoming protocol messages. We have implemented the CSA functionality as a component placed between the node and adapter components in the overlay socket. In the context of HyperCast, such a component is referred to as *processor*.

The CSA processor is constructed as a layer between the node and the adapter. To the node, the CSA processor appears as the adapter while to the adapter, it appears as the node. The CSA processor implements all address exchange methods. All incoming and outgoing protocol messages are passed through the CSA processor. Furthermore,

all address resolution requests by the node pass through the CSA processor, which gives the CSA processor control of the selection of the substrate addresses associated with a logical address.

Figures 5.1, and 5.2, respectively, show the process of receiving an incoming message and sending an outgoing message.

In Figure 5.1, an incoming message arrives to the adapter as a byte array. The adapter then passes this byte array to the *restoreMessage* method of the CSA processor where the message is inspected. If the incoming byte array is not of type CSA protocol then it is passed to the node for further processing. In this case, the node returns the resulted message to the CSA processor, which is then returned to the adapter to be buffered. Otherwise, if the message is of type CSA protocol, it will be processed into a CSA message by the CSA processor and returned to the adapter to be buffered. Messages in the adapter buffer are removed one by one and passed to the *messageArrivedFromAdapter* method of the CSA processor. The CSA processor processes CSA messages and passes all other messages to the node. Note that, as it is described later, it is possible for a CSA message to encapsulate a protocol message. In such a case, the encapsulated protocol message is passed to the node.

In Figure 5.2, the node wants to send out a protocol message. The node passes the outgoing message to the *sendMessage* method of the CSA processor. The CSA processor may choose to send this message as a payload of a CSA message. Otherwise it passes the message to the *sendMessage* method of the adapter. The adapter translates the outgoing message to a byte array and sends it out. Note that, the CSA processor is transparent to both the node and the adapter.

The CSA processor function is divided into several modules, each responsible for adding one of the address exchange methods. The CSA processor is configured as part of the overlay socket, and dependent on its configuration, various combination of address exchange methods can be enabled.

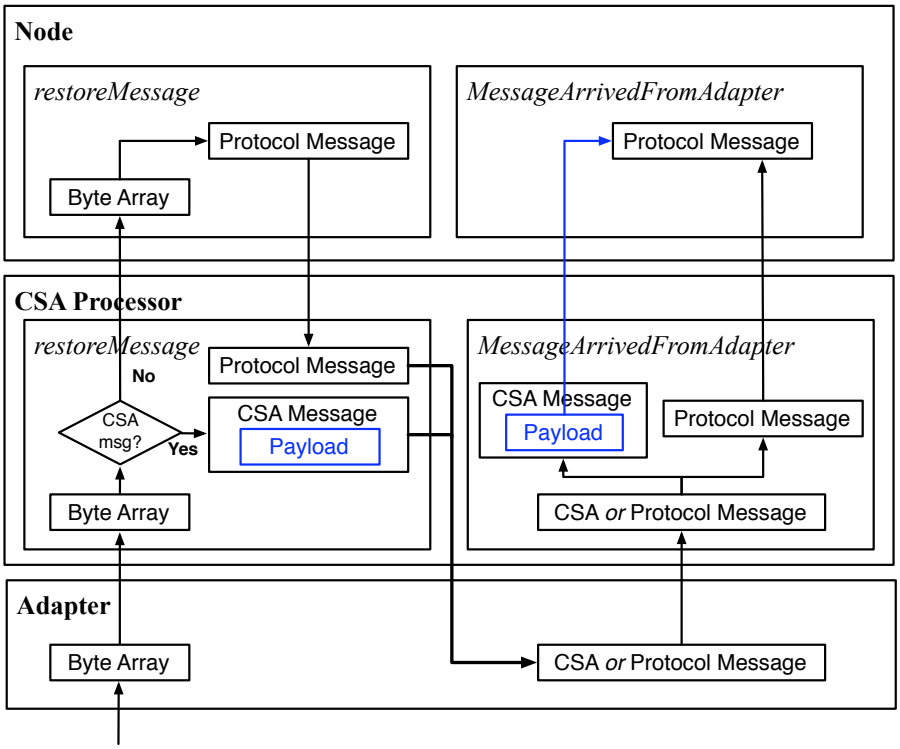


Figure 5.1: Processing an incoming message.

The operation of the direct address exchange methods is generally triggered by incoming or outgoing messages. The operation of the request method is triggered by incoming unicast protocol messages, whereas operation of the broadcast method is triggered by outgoing broadcast protocol messages and a periodic timeout event (Timers are explained in Section 5.3.2). The operation of the third party address exchange methods is not triggered by incoming or outgoing protocol messages but rather whenever the node resolves a logical address. For example, the operation of the push, pull, and gossiping methods is triggered whenever a node resolves a logical address. The gossiping method is also periodically activated by a timer.

A node queries the address repository to resolve a logical address either to use the resulting substrate address information to send a message to the owner of the logical address or to include the result in the payload of an outgoing message in order to advertise

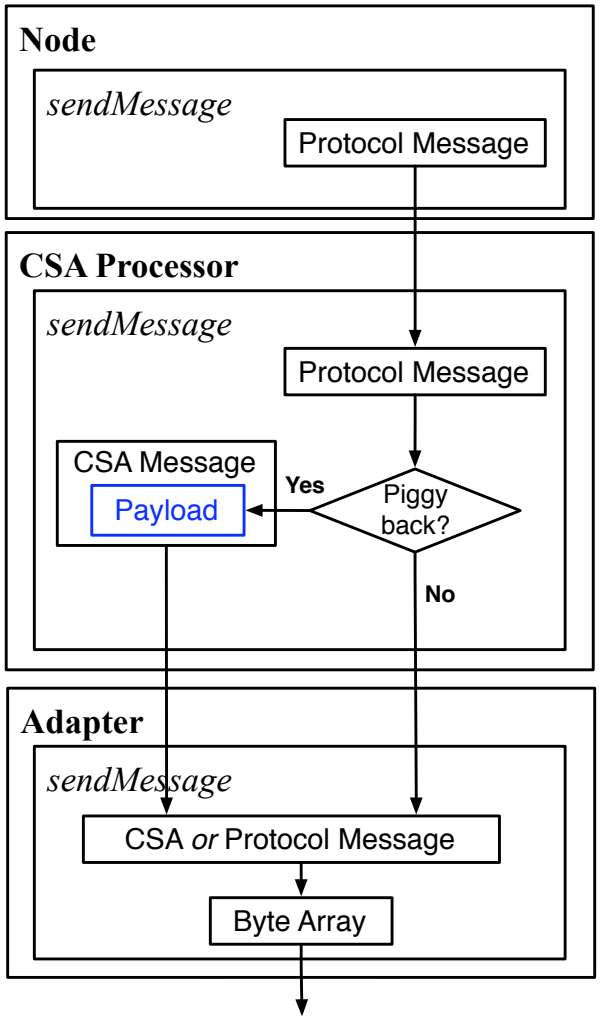


Figure 5.2: Processing an outgoing message.

the owner of that address. Since the CSA processor is between the node and the adapter, it can intercept such resolution requests and modify the result.

5.2 Implementation of Address Exchange Methods

In this section we describe the implementation of all address exchange methods. We also include example scenarios that demonstrate how address exchange is carried out for each method.

5.2.1 Request Method

The operation of the request method is triggered by the arrival of any protocol message. The request method checks if the address list of the sender is available in the address repository and whether it has recently been updated. Since the CSA processor does not have access to the logical address of the sender, the address list of the sender is identified as any address list that includes the source substrate address of the incoming message. Note that the look up of an address list with a substrate address as key is efficient because the entries in the address repository are indexed by both logical addresses and substrate addresses. If no such address list exists or if the address list is too old, a request is sent to the sender of the message requesting the complete address list of the sender. This is done by sending an `AddressListRequest` message to the source substrate address of the incoming message.

If the incoming message is a request for the address list of the local node (i.e., an `AddressListRequest` message), the request method replies with a message that contains its address list (i.e., an `AddressListUpdate` message). Figure 5.3 shows the order of messages that are exchanged and Figure 5.4 includes the flow-chart of the request method algorithm.

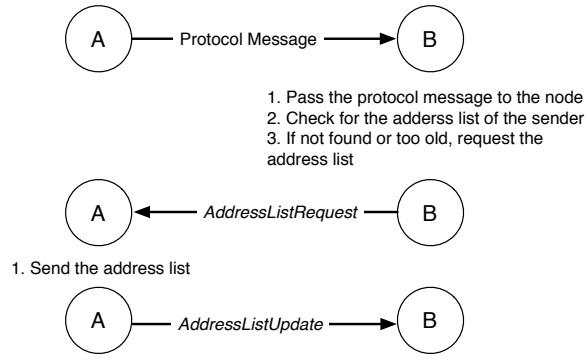


Figure 5.3: CSA messages exchanged in a request address exchange.

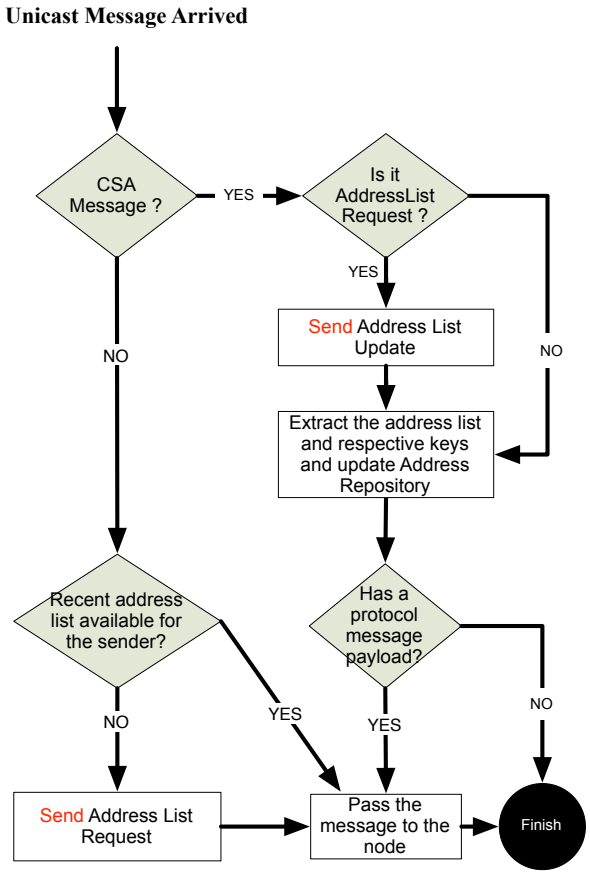


Figure 5.4: Request method algorithm.

Example Scenario

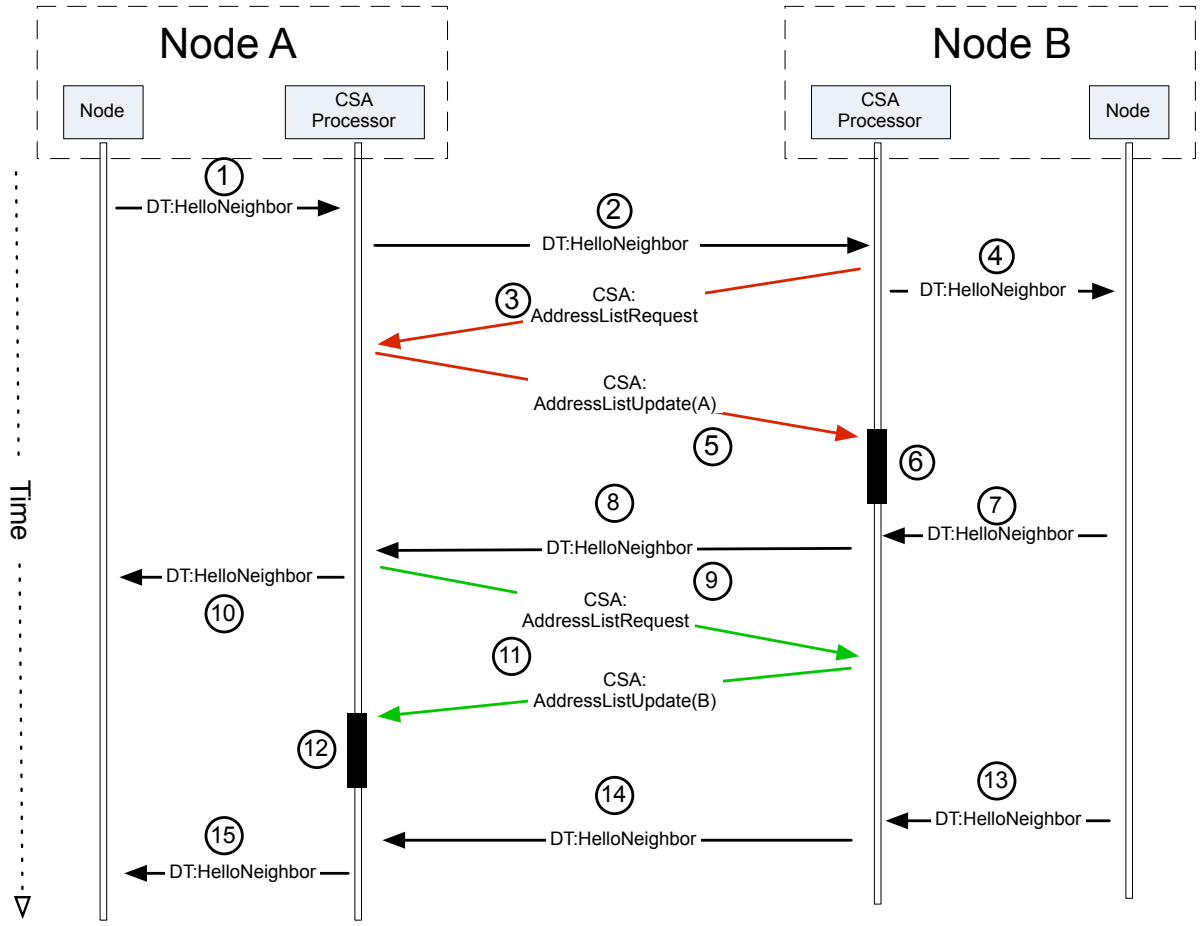


Figure 5.5: CSA message exchanged between two overlay sockets.

Figure 5.5 shows an example scenario that depicts the operation of the request method in the CSA processor. The example shows the interaction of two overlay sockets with request exchange method enabled. The overlay protocol of the sockets is assumed to be the Delaunay Triangulation (DT).

The following is the description of each message and the reason for its transmission. Protocol messages that belong to the Delaunay Triangulation (DT) protocol are prefixed with "*DT:*" and CSA messages are prefixed with "*CSA:*".

1. Node A sends a `DT:HelloNeighbor` message to Node B.

2. CSA Processor of A passes the message to the adapter which is then sent to the network.

3. CSA Processor of B receives the `DT:HelloNeighbor` message. It notices that it does not have the address list for the sender of this message and sends an `CSA:AddressListRequest` to A.

4. `DT:HelloNeighbor` message from A is delivered to node B by the CSA Processor

5. At A, the CSA Processor receives the `CSA:AddressListRequest` message from B and responds by sending an `CSA:AddressListUpdate` message to B.

6. CSA Processor of B receives the `CSA:AddressListUpdate` message from CSA Processor of A and updates its address repository.

7. Node B sends a `DT:HelloNeighbor` to node A.

8. CSA Processor of B passes the message to the adapter, which is then sent to the network.

9. CSA Processor of A receives the `DT:HelloNeighbor` message and notices that it does not have the address list of the sender in its repository. So, it requests the address list of its sender by sending an `CSA:AddressListRequest` message to B.

10. CSA Processor of A continues by passing the `DT:HelloNeighbor` message to node A.

11. CSA Processor of B receives the `CSA:AddressListRequest` message from A's CSA Processor and replies by sending an `CSA:AddressListUpdate` message to A.

12. At A, CSA Processor receives the `CSA:AddressListUpdate` message from CSA Processor at B and updates its address repository.

13. Node B sends a `DT:HelloNeighbor` message to A.

14. CSA Processor of B passes the message to the adapter which is then sent to the network.

15. CSA Processor of A receives the message. Since it already has a recent copy of the address list of the sender of the message, it will pass the message to node A without any additional operation.

Operations

Table 5.1 contains the list of events and the actions that they trigger in the request method.

[4] An entry is considered old if the node has not received an update for it for at least one minute. The exact amount of time is a configuration parameter but the default value is one minute.

[5] Key is usually a logical address. For exact definition of "key" refer to Section 5.3.1.

Table 5.1: Operations of the request method.

Event	Action

Receive a unicast protocol message `m` from a node	**if** There is no entry in the address repository for `m.source_address` OR the entry is too old[4] **then**
	Send an `AddressListRequest` message to the source.
	end if
	Pass the message to the node.

Receive an `AddressListUpdate` message `m`	Update the address repository:
	for <key[5] `K`, address list `A`> in `m` **do**
	if There is already an address list for `K` in the address repository **then**
	Update the address lists with `A` and update its timestamp.
	else
	Add `A` to the address repository under key `K`.
	end if
	end for

Receives an `AddressListRequest` message `m`	Send an `AddressListUpdate` message to `m.source_address`

5.2.2 Broadcast Method

The request method is not efficient for protocols that use multicast communication. To avoid the cost of exchanging multiple message in request address exchanges, the broadcast method relies on pushing the address lists on broadcast substrates and preventing other nodes from requesting them.

Figure 5.6 shows a flow chart of the broadcast method. The first time that a node broadcasts a message on a substrate, the CSA processor piggybacks the address list on that message. Every time the broadcast method sends out the address list on a broadcast substrate, it schedules to send the address list on that substrate again in *BroadcastPeriod*. If there is any outgoing broadcast message on that substrate within at most *AcceptableBroadcastLag* of the upcoming scheduled broadcast time, the broadcast method piggybacks the address list on it and resets the scheduled broadcast time. This is an attempt to reduce the number of broadcast messages.

The above method for broadcasting the address list ensures that the node broadcasts its address list only on substrates which have been used at least once by the node. Additionally, it ensure that the time between two consecutive broadcasts does not exceed the *BroadcastPeriod*[6] value defined in the configuration file. The *AcceptableBroadcastLag* determines the amount of time that the broadcast method is allowed to fall behind a periodic schedule to take advantage of piggybacking opportunities. The actual time between the consecutive broadcasts on each multicast interface depends on the outgoing broadcast traffic of that interface but it is bounded as follows:

$$\textit{BroadcastPeriod} - \textit{AcceptableBroadcastLag} < \text{Actual CSA broadcast period} < \textit{BroadcastPeriod}$$

The frequency of sending broadcast messages should be large enough to avoid a timeout of the node's entry in the address repositories of its neighbors. At the same time, it

[6]The default value is 5000 ms.

should be small enough to avoid generating a large amount of traffic.

Piggy-back Variation: In this variation, the broadcast method piggybacks the address list on all outgoing broadcast messages. Thus, there is no need to keep a schedule for periodical broadcast of the address list. If a node uses broadcast messages very often this method can become costly but if the node's usage of broadcast messages is sparse this variation is more efficient than a periodic broadcast.

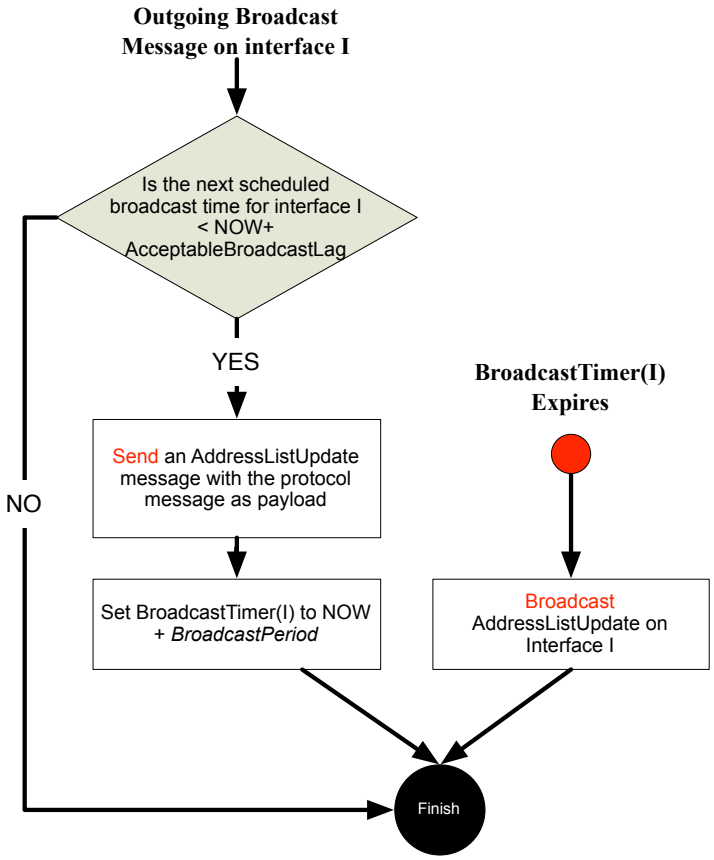


Figure 5.6: Broadcast method algorithm.

Example Scenario

Figure 5.7 depicts an example scenario that shows how the broadcast method operates. It shows the interaction of two overlay sockets that have the broadcast method enabled in their CSA processor. The overlay protocol of the sockets is the spanning tree (SPT) pro-

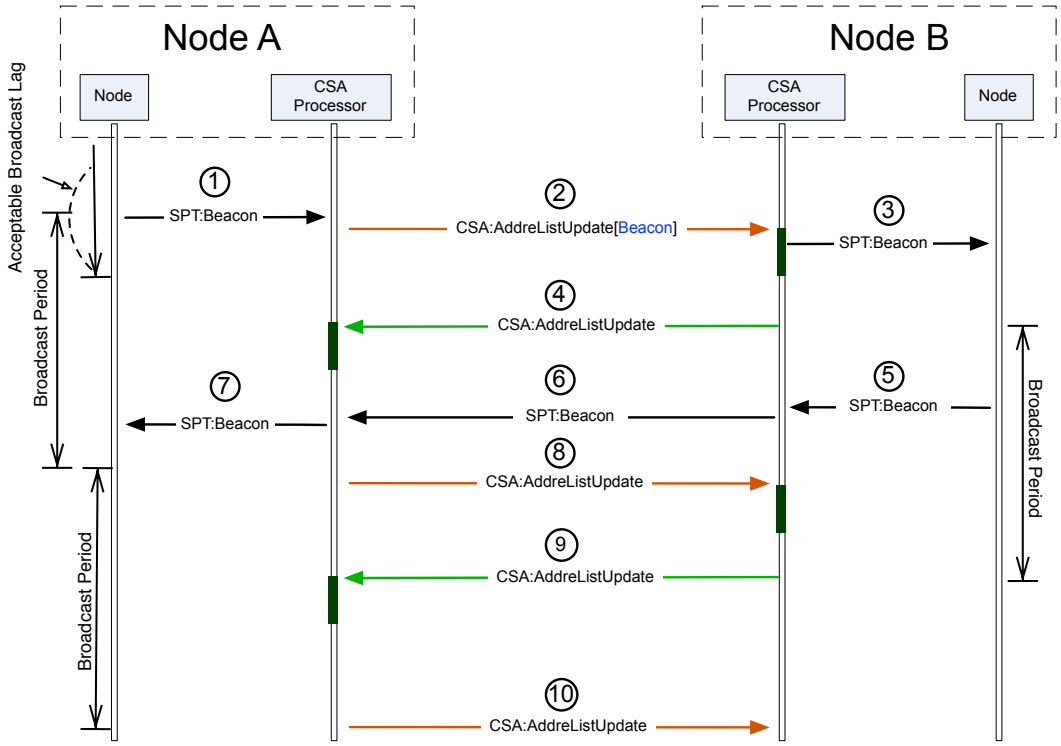


Figure 5.7: CSA message exchange for two overlay sockets on a broadcast substrate.

tocol that uses broadcast messages for rendezvous and topology maintenance. Protocol messages that belong to the SPT protocol are prefixed with "*SPT:*" and CSA messages are prefixed with "*CSA:*". The following is the description of each message.

1. Node A broadcasts a `SPT:Beacon` message.

2. CSA Processor at A notices that the outgoing broadcast message is close enough (within *`AcceptableBroadcastLag`*) to the next scheduled broadcast for the `CSA:AddressListUpdate` on this particular interface. CSA Processor at A takes advantage of the opportunity and piggybacks the `CSA:AddressListUpdate` on the outgoing broadcast message. It also resets the timer for the next schedule of address list broadcast on this interface.

3. CSA Processor at B receives the `CSA:AddressListUpdate` message and updates the address repository of B with the latest address list from A. It then passes the

payload of the `CSA:AddressListUpdate` which is a `SPT:Beacon` message to B.

4. CSA Processor at B broadcasts its address list on this substrate at the scheduled broadcast time.

5. Node B broadcasts a Beacon message.

6. CSA Processor at B receives the message, but since it is not close to its next scheduled broadcast it does not attach an `CSA:AddressListUpdate`. CSA Processor at B passes the beacon to the adapter, which is then broadcast on the substrate.

7. CSA Processor at A receives the `SPT:Beacon` from B and passes it to A.

8. CSA Processor at A broadcasts its address list on the next scheduled broadcast time.

9. CSA Processor at B broadcasts its address list on the next scheduled broadcast time.

10. CSA Processor at A broadcasts its address list on the next scheduled broadcast time.

Operations

Table 5.2 includes events and actions of the broadcast method.

5.2.3 Push Method

In the push method, whenever a protocol message contains a node advertisement, the entire address list of the advertised node is included in the message.

In HyperCast, the node resolves a logical address for two purposes: first, to send a message to the node identified by that logical address, and second, to include the result in a message for node advertisement which is the logical address of the node with

Table 5.2: Operations of the broadcast method.

Event	Action

There is an outgoing broadcast message `m` on interface `I`	**if** There is no *BroadcastTimer* scheduled for interface I (i.e., first broadcast on that interface) **or** 0 < next scheduled broadcast for interface `I` - *AcceptableBroadcastLag* < *now* **then**
	Piggyback the address list on this protocol message.
	Remove the current *BroadcastTimer(I)* if exist
	Schedule a new *BroadcastTimer(I)* event for [now + *BroadcastPeriod*]
	end if

BroadcastTimer for interface `I` expires	Broadcast `AddressListUpdate` message through interface `I` Schedule a new *BroadcastTimer(I)* event for [now + *BroadcastPeriod*]

its substrate address information. In both cases, the node obtains the substrate address information of a node through a query to the *address repository*. Since the CSA processor can intercept such queries it can return different results for different third-party address exchange methods.

For the first case, only one of the substrate addresses (i.e., the preferred substrate address) is needed, while for the second case, the complete address list is needed. To deal with both cases without introducing separate resolution methods, the CSA processor implements the resolution method that is otherwise implemented by the address repository of the adapter and returns a tuple. This problem exist for the pull method too, and a similar solution is used.

Figure 5.8 shows how CSA modifies the address resolution process for the push and pull methods. When push or pull methods are not used, the resolution method return a usable substrate address that has the highest preference value (i.e., preferred substrate address). For the push method, the resolution method returns a tuple that contains two sets of values as the result of a logical address resolution. The sets contain both the preferred substrate address and the complete address list for the resolved logical address. For the pull method, the resolution method returns a tuple that contains two sets of values as the result of a logical address resolution. However in this case, the sets contain the preferred substrate address and the logical address of the local node. The discussion regarding how this tuple is used in pull method can be found in Section 5.2.4.

When such a tuple is included in a message for node advertisement, its complete address list part is used. When this tuple is presented as a destination for a message, its preferred substrate address part is used as the destination address. All of this is transparent to the node, and this tuple is treated by the node like any other address.

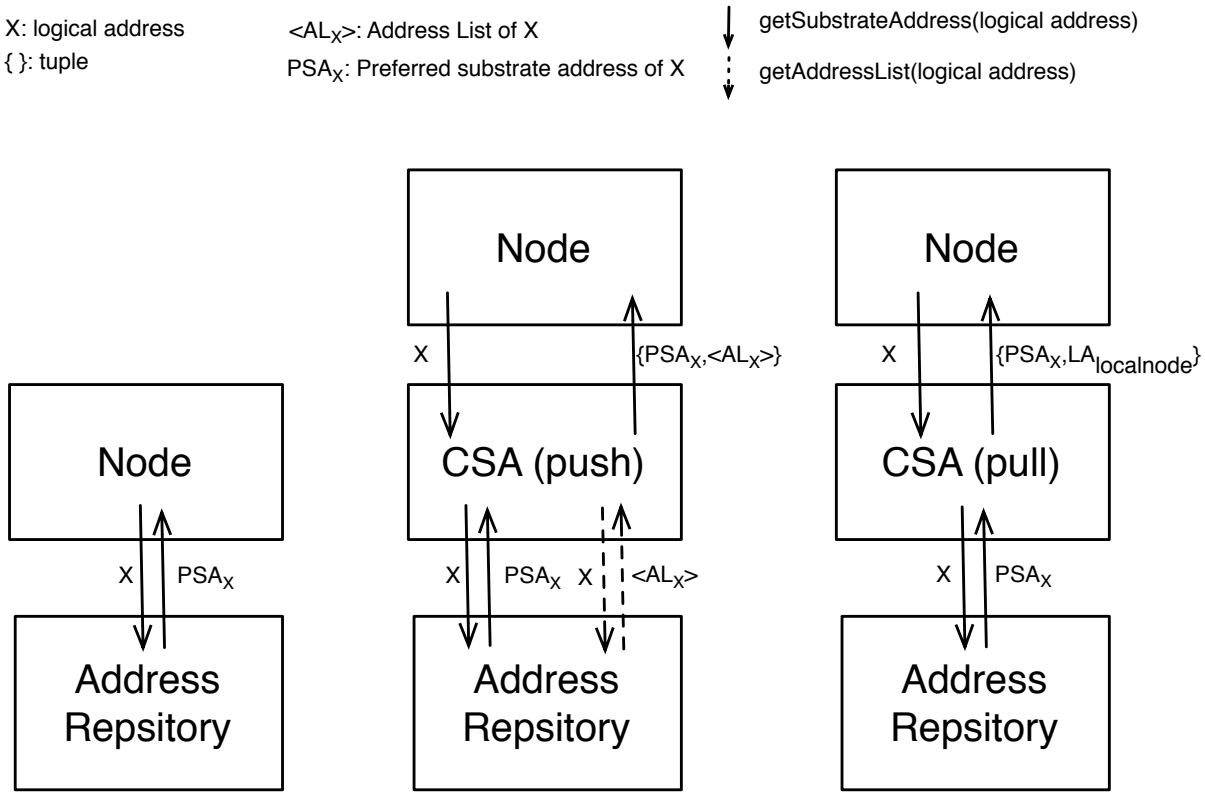


Figure 5.8: The modification of the address resolution process due to CSA.

Example Scenario

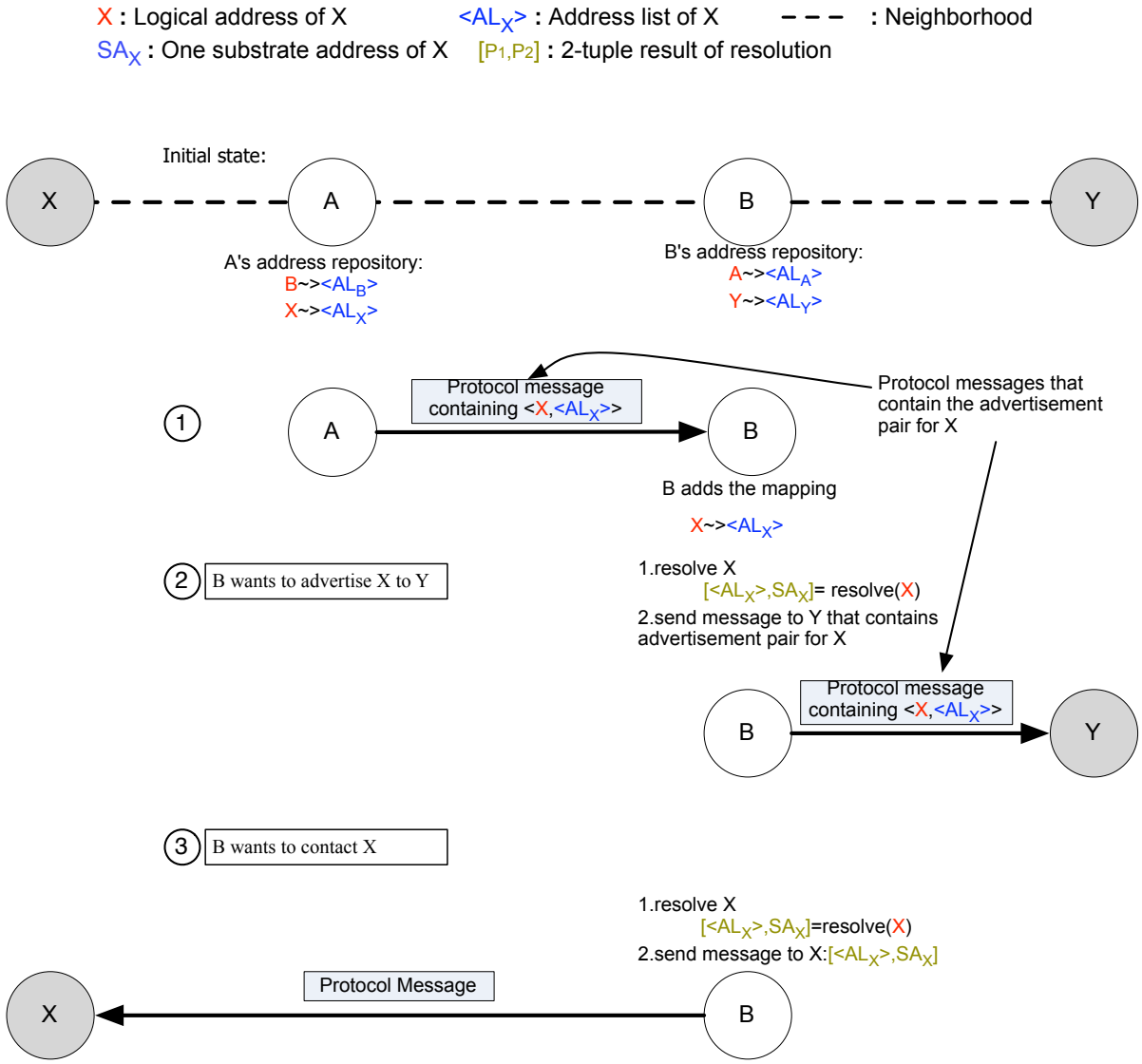


Figure 5.9: Example of push method operation.

Figure 5.9 shows an example of push method in operation. The following is the description of each part in this example.

Initial state Nodes A and B are neighbors, therefore both already have the address lists of each other in their repository. Node A is also a neighbor of node X, and thus has the address list of X in its address repository. Similarly, Node B is a neighbor

of node Y and has the address list of Y in its address repository.

1 As part of the overlay protocol, A wants to advertise X to B. A resolves the logical address of X in order to send both the logical address and the entire list of substrate addresses of X to Y as a node advertisement. The result of this resolution is a tuple that contains the address list and the preferred substrate address of X. Since, in the push method, only the address list part of this tuple is included in the messages, the address pair in the protocol message payload is <X,< AL_X >>. After receiving the message, B uses the advertised pair to create an entry in its address repository for node X.

2 Suppose that as part of the overlay protocol operation, B wants to advertise X to Y.

1. B resolves the logical address of X. The CSA processor in B returns the result of this resolution, which is the same as the result in 1.
2. B sends the protocol message to Y that contains an advertisement of X. Similar as in 1, the advertisement pair in the payload of this message is <X,< AL_X >>.

3 Suppose that, B wants to send a message directly to X.

1. B resolves the logical address of X to obtain a substrate address of X. The CSA processor in B returns the same result as in 1.
2. B sends a message destined to X and passes the result obtained in the previous step as the destination address. The CSA processor in B which sits between the node and the adapter, handles the send message process and uses the preferred substrate address of X which is part of the result in the previous step as the destination address for the message to X.

Operations

The push method in the CSA processor is invoked whenever the node resolves a logical address or attempts to send a message. Table 5.3 contains the list of all events and actions for this method.

Table 5.3: Operations of the push method.

Event	Action
Node resolves the logical address LA_X	Return a tuple that contains the address list associated with the LA_X in the address repository and the preferred substrate address of LA_X ,i.e.,$<AL_X,PSA_X>$[7].
Node sends a message $<AL_X,PSA_X>$	Send the message to PSA_X

5.2.4 Pull Method

The pull method initiates an address exchange process only when the address list of an advertised node is needed. In HyperCast, this translates to the moment when the node resolves the logical of the advertised node. Unlike the push method, in the pull method the entire address list of the advertised node is not included in the message. In fact, no substrate address of the advertised node is included in the message. Instead the advertiser includes the logical address of an node from which the receiver can obtain the address list of the advertised node. This node is the advertiser itself.

[7]PSA: Preferred substrate address, and AL: Address list.

Recall that a node *resolves* a logical address of another node to obtain the substrate address information for that node either before sending a message to it or for creating an advertisement pair to advertise that node in a protocol message. Similar to the push method, when the pull method is active, two different sets of information are needed in these tasks.

To deal with this, the CSA processor implements the resolution method that is otherwise implemented by the address repository of the adapter. In the case of the pull method, this resolution method returns a tuple that consists of the preferred substrate address for the logical address that is being resolved and the *logical address of the local node* (i.e., the advertiser). When this tuple is used as the destination for a message, its first part is put to use. The second part is used when the tuple is included in a protocol message to advertise a node.

Upon receiving a message that contains an advertisement pair, the destination node will create an entry with the following form in its address repository:

<Logical address of the advertised node ⇒ Logical address of the advertiser node >

When the destination node resolves the logical address of the advertised node for the first time, the pull method requests its address list from its advertiser. As discussed before, a tuple is returned as the result of this resolution. This tuple can be included in a message for advertisement but if the node uses it as a destination address for an outgoing message, the CSA processor puts the message in a *waiting list* until its corresponding address list which has been requested arrives. In case that the node does not receive the address list after a long time, i.e., *BufferedMessagesTimeoutTimer*[8], the entry times out and the message is dropped.

The push method is simpler than the pull method because the push method does not need to keep track of the advertiser for each advertised node.

[8]The default value is 5 seconds.

Example scenario

An example of the pull method operation is shown in Figure 5.10. The example involves four nodes: A, B, X, and Y. A and B exchange the address list of X after X is advertised by A to B. The following is a detailed description of this example.

Initial state Nodes A and B are neighbors, and both have each other's address lists. Node A is a neighbor of node X, and it has the address list of X in its address repository. Also, B is a neighbor of node Y, and it has the address list of Y in its address repository.

1 Suppose that as part of the operation of the overlay protocol, A wants to advertise X to B.

1. A resolves the logical address of X in order to create an advertisement pair and include it in the protocol message payload. The result of this resolution is a tuple that contains the the preferred substrate address of X and logical address of A. When this tuple is used for advertisement, its first part is included in the message. Therefore the address pair carried in the advertisement message is <X logical address, A logical address>.
2. A sends a protocol message containing the node advertisement pair <X logical address, A logical address> to B.
3. After receiving the message, B inserts this address pair in its address repository.

2 B wants to advertise X to Y. B resolves X. As a result of this resolution, the CSA processor in B returns a tuple that contains `NULL` as the substrate address for X since B does not yet have the address list of X in its address repository, and the logical address of B as the advertiser.

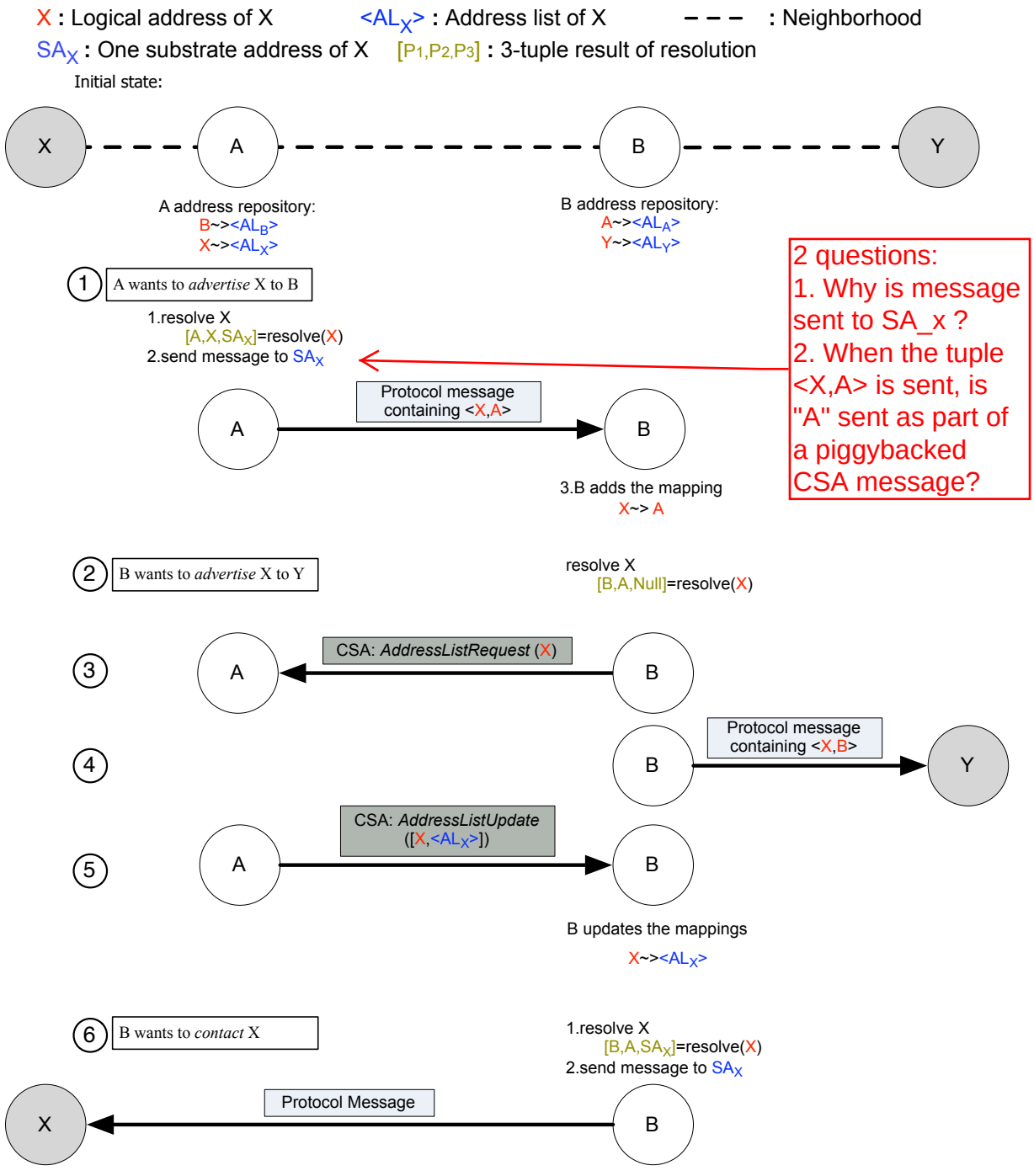


Figure 5.10: Example of pull method operation.

3 During the address resolution process in the previous step, the CSA processor in B realizes that the actual address list for X is not available in the address repository; therefore it requests the address list of X from its advertiser, A, by sending an `AddressListRequest` message to A. (B knows A is the advertiser because its logical address is mapped under the X logical address instead of the address list of X)

4 B advertises X to Y. Similar to 1, the advertisement pair in the protocol message is <X logical address, B logical address>.

5 B receives the `AddressListUpdate` message from A that contains the X address list. B updates the entry for X in the address repository.

6 Suppose that B wants to send a message directly to X.

1. B resolves the logical address of X. The CSA processor in B returns a tuple that contains the preferred substrate address of X, and the logical address of B.
2. B sends a message destined to this tuple. The CSA processor in B, uses the preferred substrate address of X from this tuple to send the message to X.

Note that in this example, if B wants to send a message to X at any stage before 5 (i.e., before it receives X address list), the message is put in a *waiting list* until B receives the X address list from A. However, B still could advertise X because the node advertisement in the pull method does not require the address list of X to be available as no substrate address of X is transmitted in the node advertisement with this method.

Operations

The pull method is used when a node resolves a logical address and upon an incoming `AddressListUpdate` messages. Table 5.4 contain events and their respective actions for the pull method.

Table 5.4: Operations of the pull method.

Event	Action

Node resolves the logical address LA_X	**if** LA_X is mapped to another logical address, i.e., the logical address of its advertiser **then**
	Send a `AddressListRequest` message to its advertiser and request the address list for the key=LA_X.
	Return a tuple that contains NULL for the substrate address, and the logical address of the local node as the new advertiser. (<NULL,$LA_{localnode}$>)
	else
	Return a tuple that contains the preferred substrate address and the logical address of the local node as the advertiser, LA_X.(<PSA_X,$LA_{localnode}$>)
	end if

`AddressListPullTimeout` timer expires	**for** each entry `e` in the waiting list **do**
	if now - `e.timestamp` > *`AddressListPullTimeout`*
	then
	Delete `e`. (i.e., drop the message `e.message`)
	end if
	end for

Event	Action

Node sends a message `m` to an address that is in fact a tuple $<PSA_X, LA_{advertiser}>$	**if** PSA_X is not NULL **then**
	Send the message to PSA_X
	else
	Buffer message `m` in the waiting list and wait for an address list for the key: LA_X
	end if

Receives an `AddressListUpdate` message m	**for** each <key, address list> `k`, `al` in `m` **do**
	Update the *Address Repository* entry for `k` with `al`.
	for each message `wm` in the waiting list **do**
	if `wm` is waiting for the address list of the key `k` **then**
	Send out `wm` and remove it from the waiting list
	end if
	end for
	end for

5.2.5 Gossiping Method

The gossiping method of the CSA consists of a periodic exchange of address lists. Each overlay node periodically sends a *gossip* message to another node which contains one or more address lists. The destination of gossip messages and address lists that they contain are selected from entries in the address repository.

In the gossiping method, when a node advertises another node in its protocol message, no substrate address of that node is included in the message payload. Instead, nodes rely on the gossiping process to provide the address lists of the advertised nodes.

When sending a message, a node resolves the logical address of the destination node to obtain its substrate address. If the address list of this node has already been obtained via the gossiping process, the preferred destination substrate address is known and returned to the node, which is then used to send the message. On the other hand, if the address list is not available in the address repository, the returned value will have an empty value for the preferred substrate address. When the node uses this returned value as a destination, the message is dropped.

Note that this is different from the pull method approach in which the message is buffered until the address list is obtained. This is because in the pull method the amount of time which the node has to wait until it receives the address is usually small (in the order of the average round trip time between two nodes), while this is not the case for the gossiping method. Having to wait a long time in the waiting list means a very large number of buffered messages which have mostly become useless given how old they are. Due to the above reasons, we just drop such messages in the gossiping method.

Example scenario

Figure 5.11 shows an example of the gossiping method in operation. The following is the description of the example in Figure 5.11.

Initial state Node A and B are neighbors and have each other's address list in their address repository. Additionally, Node A is neighbor of node X, and it has the address list of X in its address repository.

1 A wants to advertise X to B. A resolves X's logical address in order to pair the result with the logical address of X and to include it in the advertisement message payload. When the result of the resolution is included in the message, no substrate address will be included, thus the advertisement pair in the message is <X logical address, NULL>. After receiving the message, B inserts an empty mapping for X in its address repository.

2 B wants to send a message directly to X.

1. B resolves the logical address of X. The result is empty as B has not received the address list for X yet. B's node uses this result as the destination for the message.
2. The CSA processor at B gets the outgoing message destined to an empty address. The CSA processor drops the outgoing message.

Gossip Suppose that node U which is an overlay node that has address lists of X and B in its repository has randomly chosen B as its destination for gossiping. U has also randomly chosen the address list of X to gossip. U sends a gossip message to B containing the logical address of X and the address list of X. After receiving the gossip message, B updates the corresponding entry in the address repository. By doing so, the received address list of X will replace the already existing empty mapping.

3 B wants to advertise X to Y.

1. B resolves the logical address of X. Similar to step 1, an address pair <X logical address, NULL> is created.
2. B resolves Y to obtain a substrate address of Y to which the protocol message can be sent.
3. B sends the protocol message that contains the advertisement pair for X to Y.
4. Y receives the advertised address pair and creates an empty entry in its address repository.

4 B wants to send a message directly to X (as in step 2). However this time the address resolution is successful and the message is successfully sent.

1. B resolves the logical address of X. Since the address list of X is now available, the result is the preferred substrate address of the X.

2. The protocol message is sent to X.

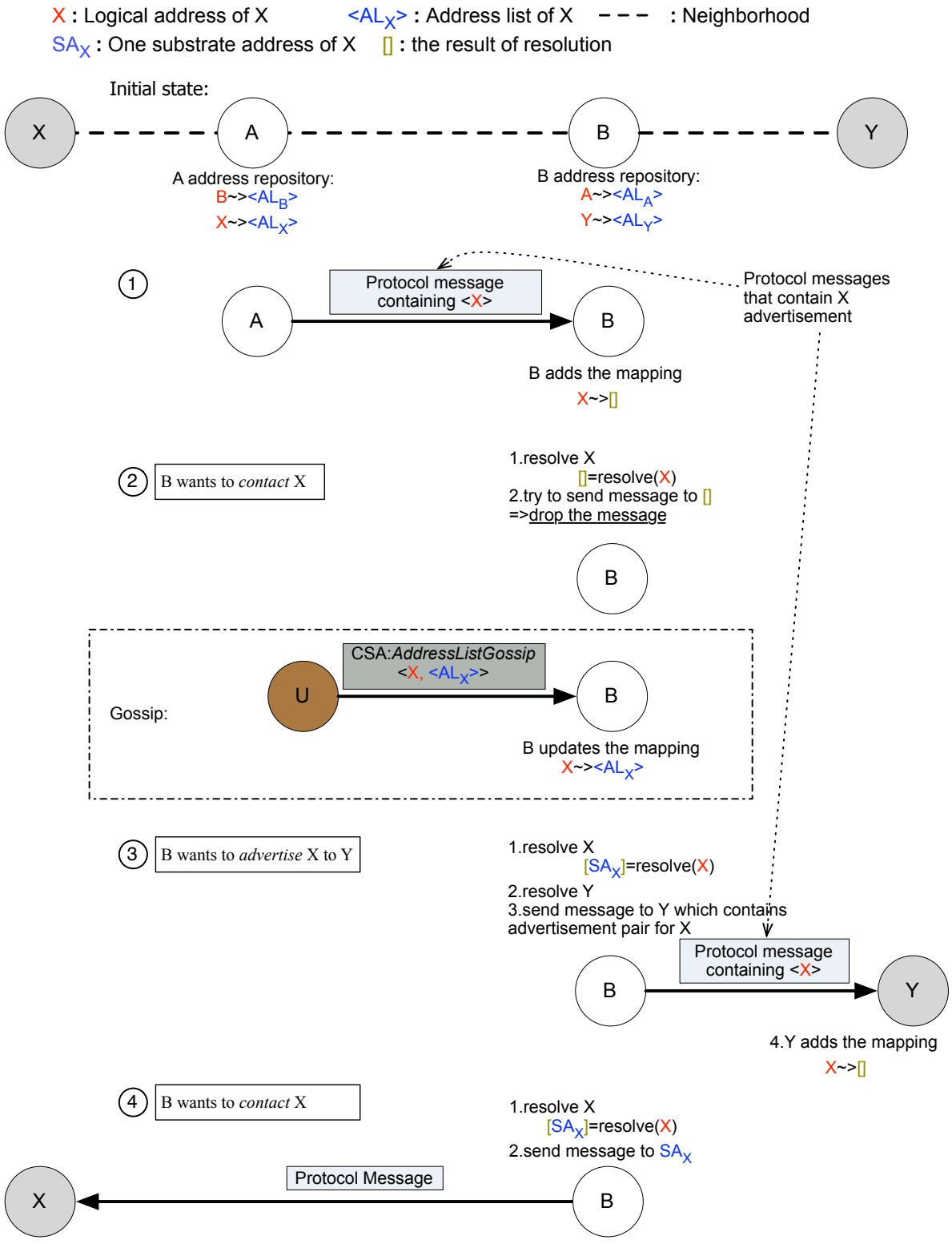


Figure 5.11: Example of pull method operation.

Gossiping in the Literature

Gossiping addresses is an information dissemination problem where a piece of information, originally known by a single node is spread to a set of nodes. The optimal solution of this problem with a deterministic series of message exchanges which requires the least number of message exchanges and gossiping rounds have been extensively studied for various types of graphs e.g. complete graphs, grid graphs, and hypergraphs [19].

Gossiping schemes with random message exchanges for information dissemination have been used in computer networks [21]. Such randomized gossiping provides a simple, robust, and scalable alternative to more rigid deterministic communication structures. For example, it has been shown that in a network with n nodes, a uniform gossiping scheme can spread a piece of information to all nodes in the network in $O(log(n))$ rounds and with $\Theta(nlog(n))$ messages. A deterministic solution (e.g., a tree with constant degree) can similarly do so in $O(log(n))$ rounds but with only $\Theta(n)$ messages [22].

Gossiping for CSA

The objective of the gossiping method in CSA is the efficient dissemination of address lists across multiple substrates. The ideal result of this method is to have each node know the address lists of all other nodes. However in large overlay network, it is not possible for nodes to store a complete set of address lists.

When some nodes in an overlay protocol may have a special role and are advertised and contacted more often than other nodes (e.g., backbone nodes), the gossiping process should give priority to the dissemination of the address lists of these special nodes.

The following aspects of a gossiping process are important in determining its effectiveness.

- **Timing**

  The gossiping process is carried out in rounds. Gossiping messages are sent periodically and independent of protocol messages.

- **Content of Gossip**

  Each node has an address repository of address lists. Each node starts with the address list of itself, and over time, it adds new address lists obtained via gossiping and other address exchange methods. The node selects address lists to gossip from this repository. Each node can explicitly express whether it wants its address list to be gossiped or not. This is indicated by flags that are associated with each gossiped address list.

- **Destination of Gossip**

  Similar to the content of the gossip, the destination for gossips is also selected from the local address repository but only from entries that are reachable, i.e., share at least one substrate with the node. The destination and the content are selected independent of each other. Finally, each node can set a flag on its address list that shows whether it wants to participate in the gossiping process as a destination for gossips or not.

- **Time-to-live of Gossip**

  In order to have an efficient gossiping process it is important to stop circulating gossip messages after they reached all participants. If the network size is available, it is possible to calculate how many times a single gossip is required to be forwarded in order be fairly certain (i.e., with a probability close to one) that all the nodes have received it. Unfortunately this information is not easily available in peer-to-peer overlay networks. One option is to have a large fixed upper bound on the number of times that an address list can be gossiped. This solution can be expensive for small network in terms of bandwidth and number of messages and, similarly, for large network by declaring a gossip messages outdated too soon. A better solution is to use a heuristic to guess the size of network. Fortunately in our case, the address lists are being gossiped which have a one-to-one relation with nodes. We think

that it is possible, to implement an estimation function that passively looks at the number of address lists that have already been received to estimate the number of nodes in the overlay network. However, this solution is not implemented in this work.

Implementation Details

Following the above guideline we have designed the following gossiping scheme. In this scheme, in each gossiping round[9], one address list is randomly selected as the destination. The address list is selected among all address lists in the CSA processor repository which have their `ReceiveGossiping` flag set to 1. Address lists do not have equal probability of being selected and their selection probability varies depending on several factors. The selection probability of an address list is proportional to its weight which is calculated based on

$$w(i) = \begin{cases} (max_i(S_d[i]) - S_d[i]) * I[i] & \text{if node } i \text{ is reachable} \\ 0 & \text{if node } i \text{ is not reachable} \end{cases} \tag{5.1}$$

where:

1. $S_d[i]$ is number of times that the address list i has been previously selected as destination for the gossip.

2. $I[i]$ is number of interface addresses in the address list i.

This equation ensures that unreachable nodes are not selected as the destination. Additionally, it assigns more weight to address lists which have not previously been selected and is biased to send gossips to new nodes. It also gives more weight to nodes which have more interfaces and are connected to more substrates. This encourages the address list to be send across substrates.

[9]The time between two consecutive gossiping round is provided in the configuration file. The default value is 500 ms.

The probability of an address list being selected given its weight is calculated by Equation 5.2. This equation ensures that when the weight for an entry is set to 0, the probability of its selection will also be 0.

$$P(\text{address list } i \text{ is selected}) = \begin{cases} \dfrac{w(i) - min_i w(i) + 1}{\sum_{i=1}^{N} w(i) - N * (min_i w(i) + 1)} & \text{if } w(i) \neq 0 \\ 0 & \text{if } w(i) = 0 \end{cases} \quad (5.2)$$

where N is the number of all address lists in the address repository.

After selecting the destination, the gossiping method randomly selects a set of address lists from the address repository that have an `AdvertiseGossiping` flag set to 1 for inclusion in a gossiping message. The number of address lists included in a gossip message to be gossiped has an upper bound determined by a configurable parameter[10]. Similar to the destination case, the selection probability of each address list varies depending on several factors, and the respective weight of each address list is calculated by

$$w(i) = max_i(S_g[i]) - S_g[i] \quad (5.3)$$

where $S_g[i]$ is the number of times that address list i has been previously gossiped by the local node. This equation

The destination and content of gossips are both selected from the address lists in the address repository but independent of each other. Note that the effectiveness of the gossiping process in disseminating address list is sensitive to the above weight assignments and overlay topology. The gossiping process efficiency can be improved by appropriating these weights based on the overlay topology.

[10]The configuration parameter is `MaxGossipPerMsg`. (Default value is 1)

Operations

Table 5.5 contains the events that trigger an action by the gossiping method.

Table 5.5: Operations of the gossiping method.

Event	Action
GossipTimer expires	Choose a destination from the address repository. (For details refer to Section 5.2.5) Construct a gossip message by choosing address lists to be gossiped. Send the gossip message to the selected destination.
Receives an `AddressListUpdate` message `m`	Extract all address lists and update the local Address Repository.

5.2.6 Push-Single/Gossiping Hybrid Method

This method combines the gossiping method and the push method (see Section 4.2.2). In this method, the push method is used to send only one substrate address (hence the name push-single) as the substrate address information part of a node advertisement. This substrate address is the most preferred substrate address according to the preference values assigned by the owner of the address list. To receive the complete address list, the node relies on the gossiping method which is active in the background.

5.3 Message Format and Timers

5.3.1 Message Format

The cross-substrate advertisement protocol is implemented as a an overlay protocol in HyperCast. Therefore, its message format follows the convention of any protocol messages in Hypercast: (1) the first byte identifies the protocol, (2) the next two bytes define the total length of the message, and (3) the next byte specifies the protocol specific message type. All cross-substrate advertisement messages have common header fields as shown in Figure 5.12. These fields are referred to as the `CSAdvertisementExchange` header. The remainder of the message is different depending on the message type.

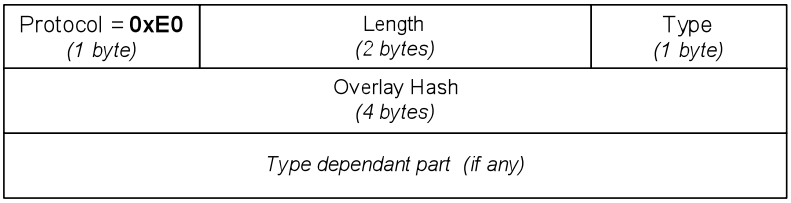


Figure 5.12: CSA message format.

The following message types are defined for the CSA protocol:

1. Address List Request;
2. Address List Update;
3. Address List Update With Payload;
4. Address List Gossip.

Address List Request

Type: 0x00

This message is used to request an address list from another overlay socket. It carries an

address key that identifies the address list which is being requested.

The CSA processor requests address lists from other nodes in two cases: 1) A message has arrived from a node whose address list is not available or is old; 2) The pull method is used for a third-party address exchange, and an address list for an advertised node is needed. In this case a request is sent to the advertiser for the advertised node.

The address key determines which address list is requested. For example, for the pull method, the address key is the logical address of the advertised node for which the address list is being requested. The format of the address key is shown if Figure 5.13. It is possible to specify the key to be empty which means that the address list of the receiver of the message is requested. The following are all different types of the address key:

None (0x00) This identifies an empty key. No key value is carried with this type of key, hence the size of this key is 1 byte.

Logical Address (0x01) This type of key is a logical address. The key value carried is a logical address. The size of the logical address is a parameter of the overlay protocol and is known by all nodes in the overlay network. Therefore there is no need to include the length along with the logical address.

Arbitrary Size (0x02) This type of key is a byte array of arbitrary size. In this case, the first byte of the key value is the array size.

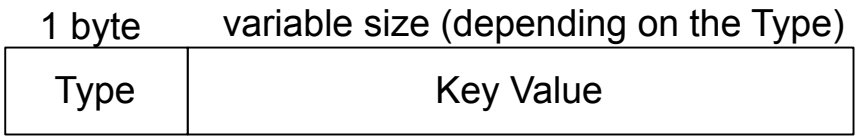


Figure 5.13: Address key format.

Address List Update

Type: 0x01

This message contains a set of address lists with their keys. It is used in two situations. First, in response to an `AddressListRequest`, in which case it contains the address list for the requested node. Second, as a broadcast message which is sent periodically in broadcast-capable substrates, where it contains the address list of the sender. Figure 5.14 shows the format of the type dependent part for this message. It includes a list of address lists and their corresponding address keys. Figure 5.15 shows the format of an address list. Each address list consists of a list of *address elements*. Figure 5.16 shows the format of the address element. Each address element consists of a substrate address and two 4 bits preference values. Section 4.1.3 contains more information about these preference values.

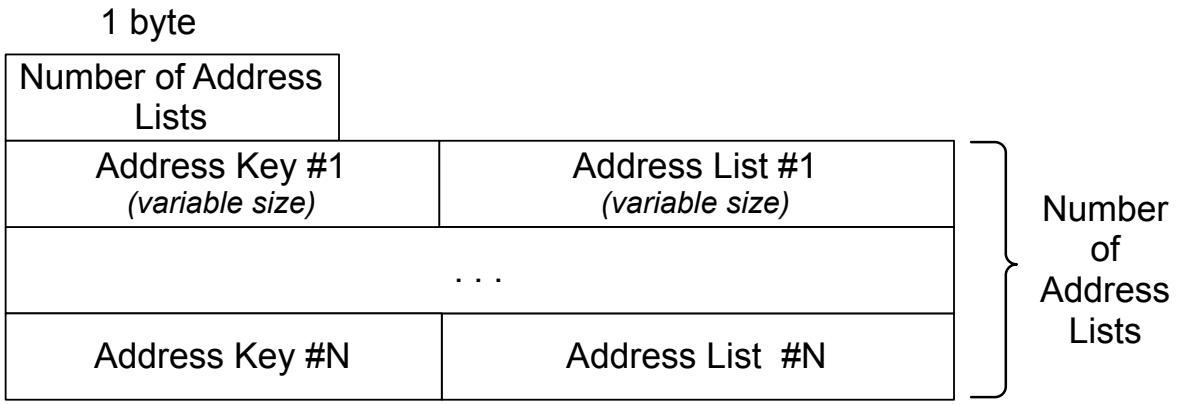


Figure 5.14: Address List Update message format.

When a CSA processor receives an `AddressListUpdate` message, it processes each address list by translating its address elements to substrate addresses. If the hash value of an address element matches the `hash(address realm, protocol type)` of an interface of the overlay socket, it belongs to that interface s substrate network. In this case, the address byte array of the address element is processed by that interface into a substrate

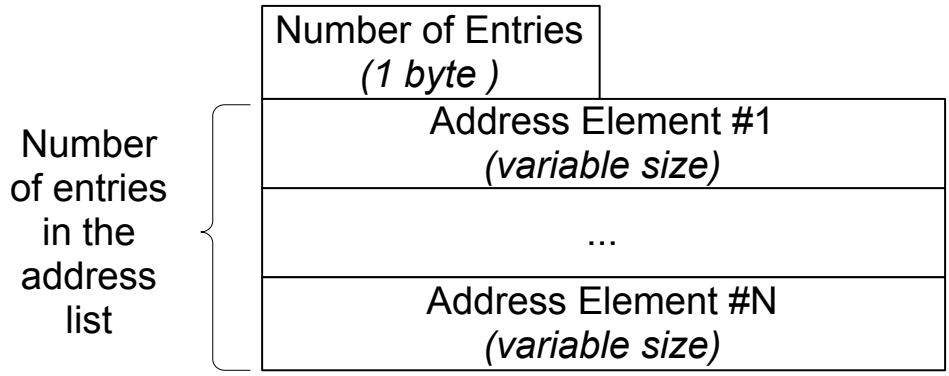


Figure 5.15: Address list format.

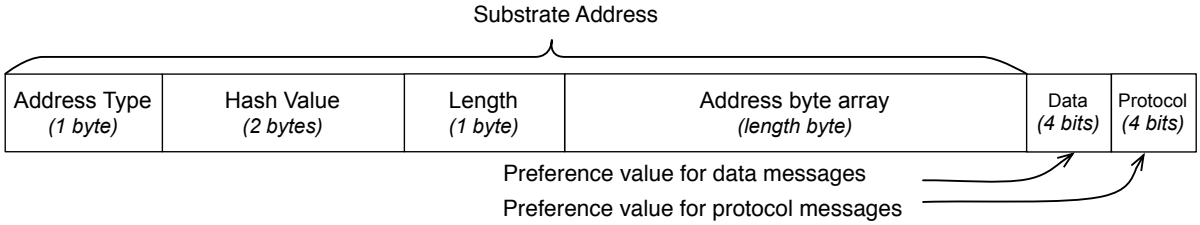


Figure 5.16: Address element format.

address.

Any address element with a hash value that does not match the hash value of any interface of the overlay socket, is kept as a byte array and tagged with a special interface ID that indicates it is not usable by this overlay socket. Although the local node cannot use such addresses, they are kept alongside the usable interface addresses as the node may later send them to other nodes in a node advertisement.

The hash function used for hashing the address realm and protocol type values is based on the hash function that is used to generate the *overlay hash* in the HyperCast. The overlay hash function returns a 4 bytes value, however we need a 2-byte value as the hash value of the address realm and the protocol type values. To address this issue, the 4-byte value returned by the overlay hash function is transformed to a two bytes value by taking the XOR between first two bytes of the 4 bytes hash value as the first byte,

and the XOR between last two bytes of the 4 bytes hash value as the second byte. With this 2-byte hash value 65536 unique substrates can be coexist.

Address List Update With Payload

Type: 0x02

This message is similar to the `AddressListUpdate` message with an additional protocol message payload included at its end. This message is primarily used to piggyback address list update message on outgoing broadcast protocol messages whenever possible.

Address List Gossip

Type: 0x03

This message is used in the gossiping method to transfer multiple address lists (up to 255). Each address list in this message has additional information associated with it. We refer to each address list along with its associated additional information as an *extended address list*. The format of the extended address list is shown in Figure 5.17. Each extended address list carries the following properties in addition to the normal address lists:

- **Gossiping Counter**: The gossiping counter is the number of nodes which has gossiped this address list before. Every time an address list is being gossiped, its gossiping counter is increased by one. The gossiping algorithm can use this value to decide whether this address list has been already gossiped enough or still needs to be gossiped. Although the current implementation of the gossiping method does not use this field, it is kept for future implementation.

- **Gossiping Flags**: Two flags, *ReceiveGossiping* and *AdvertiseGossiping* are defined as following:

  - Flag *ReceiveGossiping*(R): If this flag is set, the owner of this address list

wants to participate in the gossiping of address lists and can be selected as a destination for gossip messages. This flag is configured using the property `GossipReceiver` in the configuration file (default: true).

– Flag *AdvertiseGossiping*(A): If this flag is set, the owner of the address list wants its address lists to be gossiped. This flag is configured using the property `GossipAdvertisable` in the configuration file (default: true).

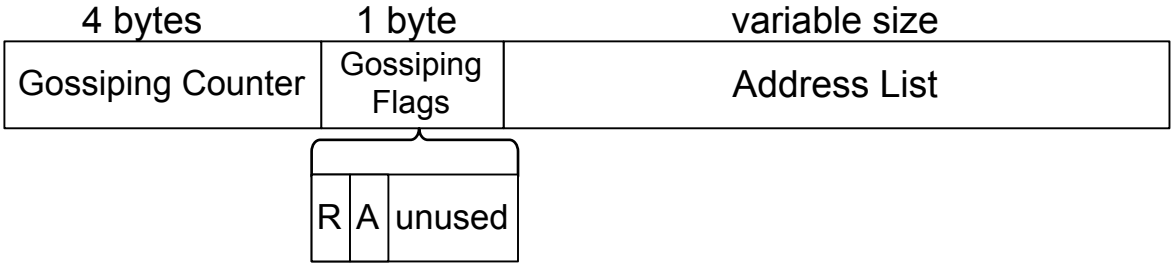


Figure 5.17: Extended address list format.

5.3.2 Timers

Timers in HyperCast are the mechanism by which components, such as the node and the CSA processor, schedule their time dependent operations. Each component can setup a timer that expires after a certain amount of time. Upon expiration, a callback is issued to handle the timer event. For creating a periodic event, the expiration callback sets the timer again. The following are the timers of the CSA processor. The timeout values of the timers are properties which are set in the configuration file.

BroadcastTimer There is one BroadcastTimer per broadcast interface of the overlay socket. The broadcast timer for a interface indicates the next time that the address list of the local node must be broadcast on that interface. Upon the timer expiration, the CSA processor broadcasts an `AddressListUpdate` message containing its

address list on the corresponding interface. The timer is reset every time it expires or whenever the address list is piggybacked on an outgoing broadcast message, i.e., when a protocol message is going out within at most *AcceptableBroadcastLag* of the expiry time of the timer.

Related values: *BroadcastPeriod* (default: 5000 ms), *AcceptableBroadcastLag* (default: 1000 ms)

GossipTimer If gossiping is enabled in the CSA processor, this timer expires periodically for each gossiping round. In each round, the node sends out one `AddressListGossip` message.

Related values: *GossipTime* (default: 2000 ms)

AddressListTimeoutTimer When this timer expires, the CSA processor checks all its address lists in the address repository and removes them if they are stale. An address list is stale if it has not been updated for *AddressListTimeout* time. This timer expires periodically every *AddressListTimeout*/2.

Related values: *AddressListTimeout* (default: 60 seconds).

AddressListRequestTimeoutTimer This timer is used when the pull method is enabled in the CSA processor. When this timer expires, the CSA processor removes any pending address list request for which no response has been received for *AddressListRequestTimeout* time. This allows the CSA to try again to send another request for the address list.

Related values: *AddressListRequestTimeout* (default: 2000 ms).

BufferedMessagesTimeoutTimer Similar to the previous timer, this timer is period-

ically active when the pull method is used. On its expiry, the CSA processor drops messages which have been waiting too long (longer than *BufferedMessagesTimeout* time) on the address list of their destination to be successfully pulled from its advertiser.

Related values: *BufferedMessagesTimeout* (default: 2000 ms).

Chapter 6

Evaluation

In this chapter, we present our evaluation of the cross-substrate advertisement mechanisms. The evaluation has two objectives. First we evaluate the overhead of the CSA mechanism proposed in this thesis. Second, we assess the effectiveness of the CSA mechanism in supporting self-organizing overlay networks in a multi-substrate environment. In particular, we study the effect of CSA on the rendezvous process of the overlay, the overlay topology, and overlay robustness.

For the evaluation, we have conducted measurement experiments on a testbed network. Before presenting the experiments, we describe the testbed network and how we create a multi-substrate environment in which our mechanisms are evaluated.

6.1 Setup of Experiments

6.1.1 Testbed Network

For our experiments we use a local testbed network configured using *Emulab*. Emulab is a software system for networked and distributed system experimentation [38]. It manages a pool of physical machines and a switching infrastructure based on VLAN technology providing a exible way to remotely congure a network configuration. Emulab has a

web-based access system to control and manage ongoing experiments.

The Emulab testbed at the University of Toronto consists of 22 Dell PowerEdge 2950 III PCs (20 test machines and 2 control servers), with 2 Quad-Core Intel Xeon Processors 5400 series clocked at 2.00 GHz, and 4GB DDR2 RAM. Each PC is equipped with an Intel VT PCIe Quad-port Copper Gigabit Ethernet NIC and a *NetFPGA* [26] board with 4 Gigabit Ethernet interfaces. So, each PC has 8 Ethernet interfaces that can be used in an experiment. The PCs are interconnected by four 48-port Cisco Catalyst 4948-10GE switches. Figure 6.1 is a photo of this testbed.

Emulab enables the creation of arbitrary network topologies and supports a wide variety of operating systems. We run our experiments on a Fedora Core 6 Linux distribution with the 2.6.20.6 Linux kernel.

Figure 6.1: The Emulab testbed at the University of Toronto.

6.1.2 Emulating Multiple Substrates

In order to be able to create many substrate networks we use UDP/IP substrate networks. Each substrate has a private IP address space and its own distinct address realm. Multicast substrates are UDP multicast/IP substrate networks. Again, each substrate has its own multicast group address and address realm. This ensures that a broadcast message in one multicast substrate cannot be received in others.

Since all substrates are UDP/IP, addresses of nodes in different substrates have the same format, i.e a pair of an IP address and a port number. However, since each substrate address includes the address realm of the substrate, addresses will only be used in their own substrate network.

6.1.3 Arrangement of Substrates for DT Protocol

For our experiments, we use a structured overlay protocol, which is based on building a Delaunay Triangulation topology. The protocol is referred to as DT protocol. For the DT protocol to form a single complete triangulation, neighboring nodes in the topology must have at least one substrate in common. Note that a DT node has an (x,y) coordinate as its logical address. Given coordinates of all nodes, it is feasible to use coordinates of a node and Delaunay Triangulation properties to identify the neighbors of one node in the topology. Since neighborhood relations are determined by the coordinates, we can ensure feasibility of a stable triangulation topology over the given substrates by carefully assigning coordinates to nodes in each substrate.

The following rules define how coordinates are assigned to nodes in substrates to ensure that the DT protocol can create a stable Delaunay triangulation over a multi-substrate environment.

1. The coordinates of all the nodes that belong to one substrate are within a confined square-shaped area referred to as *substrate field*. For example, the node in a sub-

strate whose substrate field is centered at (x_o, y_o) and with side length L will have coordinates (x, y) such that $(-\frac{L}{2}, -\frac{L}{2}) \leq (x, y) - (x_o, y_o) \leq (\frac{L}{2}, \frac{L}{2})$.

2. Substrate fields are arranged in a K-by-K two dimensional grid. For $K > 1$, each substrates is adjacent to either 3, or 8 other substrate fields depending on its position in the grid.

3. Each substrate field is divided into four equal square shaped regions.

4. The two top regions of each substrate field are shared with the top neighboring substrate field. Similarly the two bottom, two left, and two right regions are share respectively with the bottom, left, and right neighboring substrate fields. The top right region of each substrate field is also shared with the top right neighboring substrate field. Similarly, top left, bottom right, and bottom left are shared with respective diagonally neighboring substrate fields. Therefore, a single region can potentially belong to multiple substrate fields.

5. Depending on the region in which a node is located, a node is connected to one, two, or four substrates.

Figure 6.2 shows a scenario with 4 substrates and their respective substrate fields, which are arranged in a 2x2 grid based on the above mentioned rules. As can be seen, each substrate field overlaps with three other substrate fields in this example. Nodes whose coordinates fall into region $R_{1,1}$ are only connected to substrate S_1. Nodes in region $R_{2,1}$ are connected to substrates S_1 and S_3, and nodes in region $R_{2,2}$ are connected to substrates S_1, S_2, S_3, S_4.

All nodes in a region are pre-configured with the address of one special node, i.e., a "buddy", in the same region. With this buddy, nodes in the same region can rendezvous and construct the topology at least within this region. All nodes in the same region has one additional buddy which is located in a neighboring region. The neighboring regions

from which the second buddy is selected are assigned based on a specific scheme which is shown in Figure 6.3. As it can be seen in this figure, the buddies are selected such that the form a chain among regions. This scheme ensures that the least but sufficient amount information about buddy nodes is available so that nodes in different substrates can form a connected overlay network.

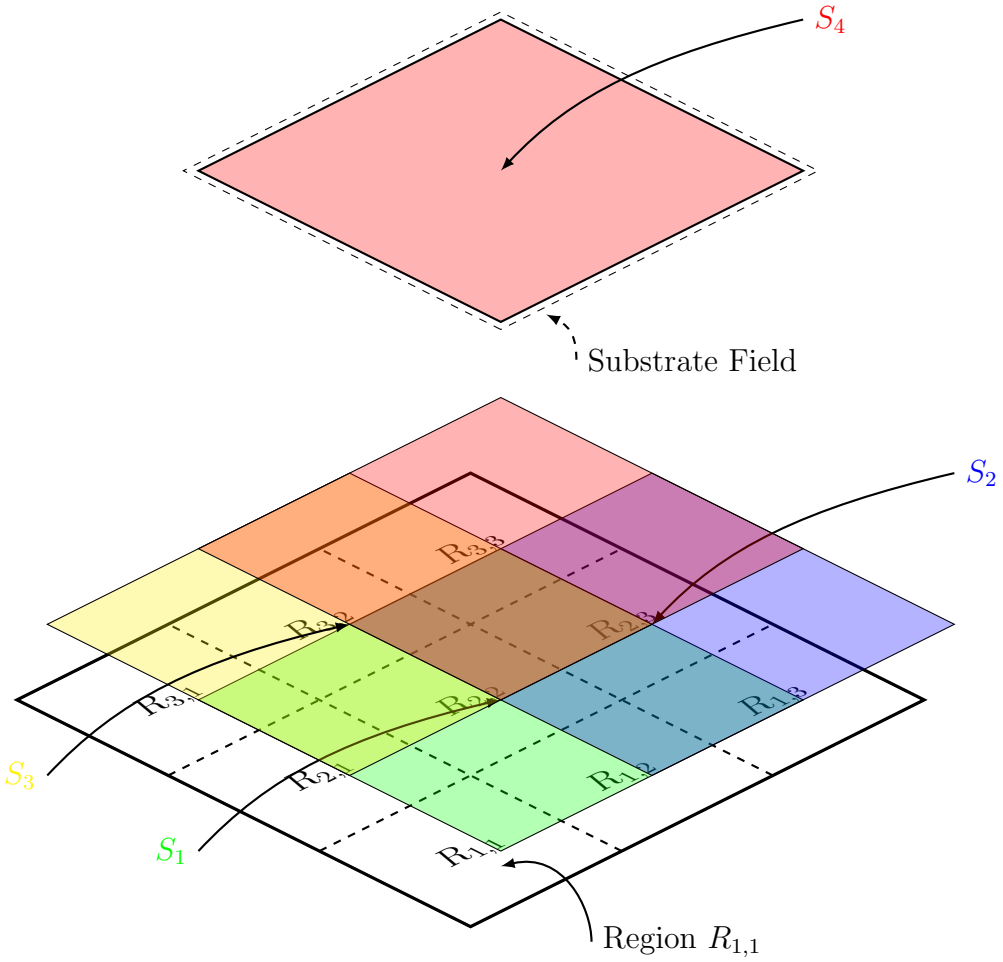


Figure 6.2: Substrate arrangement for DT.

6.1.4 Measurement Methodology

The number of overlay nodes is distributed evenly to all hosts involved in an experiment. Nodes which belong to one region are started, stopped and controlled remotely using

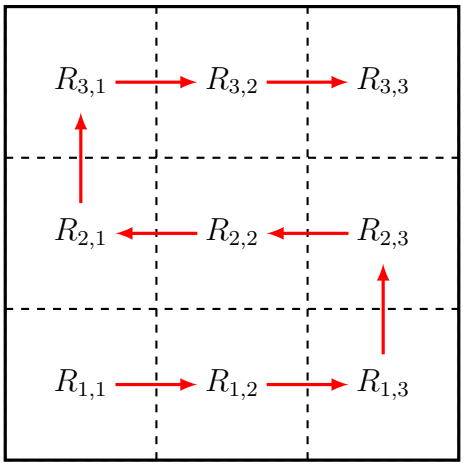


Figure 6.3: Buddy configuration for the DT protocol in a 2x2 grid. The arrows show the neighboring region from which the pre-configured buddy is selected.

the monitor and control system of HyperCast [23]. Each overlay node periodically logs parameters in a file. These files are collected at the end of an experiment for further processing.

To see the overhead incurred by the CSA, we measure the traffic generated by the CSA and compare to the traffic generated by the DT protocol. We measure the effectiveness of CSA methods by measuring their impact on overlay protocol properties such as stability and connectivity. The following are the definition of stability and connectivity that we use for the DT protocol:

Stability of DT protocol: In the context of the DT protocol, a node is stable if it has established neighborhood relation with all nodes in the overlay network that pass its neighbor test. The neighbor test can be found in [24]. If all nodes are stable, then a logical Delaunay triangulation has been established globally.

Connectivity of DT protocol: In the DT protocol, a node is a *leader* if it does not have a neighbor with a larger coordinate[11]. We say that a set of nodes is connected

[11] $coord(A) < coord(B)$, if $y_A < y_B$, or $y_A = y_B$ and $x_A < x_B$.

if there is only one leader in the overlay network. Initially, all nodes assume to be a leader, and the number of leaders decreases as the network is established. Note that a DT network maybe connected, while not being stable, while a single stable network is always connected.

The running time of an experiment is 450 seconds. This is large enough so that a DT network of the considered scale can become stable. CSA and DT parameters are logged every second.

6.1.5 Configuration Parameters

The following are relevant parameters of DT and CSA protocols:

- DT
  - Fast Heart Beat time (i.e., Heart beat when node is not stable): 500 ms.
  - Slow Heart Beat time (i.e., Heart beat when node is stable): 500 ms.[12]
- CSA
  - Maximum number of address lists per gossip message: 1 address list.
  - Gossiping table size: unlimited.
  - Address List Timeout: 60 seconds.
  - Address List Request Timeout: 2000 ms.
  - Buffered Message Timeout: 5000 ms.

6.1.6 Address Exchange Methods Used in Experiments

The following configurations have been evaluated in the experiments. The request method has been used in all of the following configurations for direct address exchange.

[12]Slow and fast heart beats are intentionally set to be equal in order to have the same timing behavior for all choices of third-party address exchange methods regardless of their effect on the stability.

- **None**: Uses no third-party address exchange method which means that no substrate address information is included in the protocol messages.
- **Push**: Uses the push method for third-party address exchanges.
- **Pull**: Uses the push method for third-party address exchanges.
- **Gossiping (250 ms)**: Uses the gossiping method with a gossiping period of 250 ms for third-party address exchanges.
- **Gossiping (500 ms)**: Uses the gossiping method with a gossiping period of 500 ms for third-party address exchanges.
- **Gossiping (1000 ms)**: Uses the gossiping method with a gossiping period of 1000 ms for third-party address exchanges.
- **Push-single**: Uses the push-single variation of the push method for third-party address exchanges which includes a single substrate address in protocol message for node advertisements.
- **Push-single & Gossiping (500 ms)**: Uses a combination of the push-single method and the gossiping method with gossiping period of 500 ms for third-party address exchanges.

The following configurations use the broadcast method as well as the request method for direct address exchanges, and the push method for third party address exchanges.

- **Periodic Broadcast (6 s)**: For the broadcast method, the *BroadcastPeriod* is set to 6 seconds and *AcceptableBrodcastLag* is set to 200 ms.
- **Periodic Broadcast (30 s)**: For the broadcast method, the *BroadcastPeriod* is set to 30 seconds and *AcceptableBroadcastLag* is set to 1000 ms.
- **Piggyback Broadcast**: Uses the piggyback variation of the broadcast method.

6.2 Experiment 1: 648 nodes in 64 substrates (8x8 grid)

In this experiments, we have used the DT protocol to study the overhead and effectiveness of cross-substrate advertisement methods. The experiments consist of unicast-only UDP/IP substrates as described in Section 6.1.2 which are arranged in a grid as described in Section 6.1.3. All substrates contain the same number of nodes. In the experiments we have evaluated different third-party address exchange methods by varying the number of substrates and the number of nodes in each region.

We first show measurements conducted on a DT network with 648 nodes. There are 64 substrates arranged in a 8x8 grid and 81 regions (in a 9x9 grid). There are 8 nodes in each region, which means that there are 32 nodes in each substrates. Each data point is the average of three runs with the same initial configuration and coordinates for nodes. 16 PC from the Emulab testbed were used for this experiment. The network configuration is that all PCs share a single 1Gbps LAN.

Figure 6.4 shows the percentage of stable nodes in the DT network as a function of time. The push and pull configurations are both highly successful in quickly establishing a stable DT network (in less than 20 seconds they achieve 100% stability). This is because both methods are good at providing each node with enough substrate address information of their neighbors so they can become stable. The none configuration achieves 0% of stability due to the fact that the node advertisements do not include any substrate address information thus nodes cannot contact potential neighbors to establish neighborhood and achieve stability.

The gossiping method takes a longer time to reach stability than the push and pull methods. The graph contains the results of the gossiping method with three configurations with gossiping time interval set to 250, 500, and 1000 ms. The speed of reaching stability is inversely related to the gossiping time interval. This is expected as a smaller

gossiping time interval means faster spread of address list information. With the push-single method, the network does not achieve complete stability. This is because, the push-single method provides only one substrate address which is not always sufficient for establishing required neighborhood relations in the configured network. The combination of the gossiping method and the push-single method is able to achieve 100% stability. This combination results in an initial jump in the number of stable nodes which is due to the effect of the push-single method. Additionally, with this method, the network becomes completely stable, because, over time, the gossiping method disseminates complete address lists which makes up for the shortcoming of the push-single method.

Figure 6.5(a) depicts how the number of leader nodes in the DT overlay network decreases over time with each third-party address exchange methods. As discussed before, a fully connected DT network has only one leader node. When no third-party address exchange is used (i.e., in the none configuration), the number of leader nodes cannot be reduced to fewer than around 250 leaders. This large number of leaders indicates the difficulty of the experimental setup. The push and pull methods enable the network to become fully connected (i.e., to one leader) in less than 20 seconds.

With all three gossiping configurations, the number of leaders decreases over time and the network becomes increasingly connected. The decrease rate of number of leaders increases with smaller gossiping intervals. Over the duration of the experiment, the gossiping method is unable to create a connected network with a single leader. For all values of the gossiping interval, the rate at which the number of leaders decreases declines over time. This is due to the fact that, over time, the efficiency of the gossiping method declines. Apparently, as the experiment progresses the address list information exchanged by nodes contains less and less new information. It may be possible to improve this issue by improving the gossiping scheme.

In Figure 6.5(b) we show the same data as Figure 6.5 for values of the number of leader less than 10. The figure shows which methods succeed in creating a single

Table 6.1: Average and standard deviation of the received CSA and DT trac for 64 substrates.

Method	CSA (bps)	stddev	DT (bps)	stddev	Total (bps)
none	33.21	53.52	2318.69	3255.67	2351.9
gossip (1000ms)	568.56	744.55	3534.30	2072.20	4102.86
gossip (500ms)	1116.03	1500.17	3815.05	1783.79	4931.08
gossip (250ms)	2173.22	2940.41	4111.36	1455.59	6284.58
push-single	69.43	32.05	6339.75	1839.02	6409.18
push-single & gossip (500ms)	928.52	915.32	6466.30	1727.30	7394.82
pull	80.77	38.37	6294.49	1813.41	6375.26
push	55.56	16.11	15060.39	5058.43	15115.95

shows the min and maximum for the value in all runs. As it can bee seen the DT protocol traffic varies with different third-party address exchange methods. This is because the size of messages which include node advertisements is different with each method as they each include a different value as the substrate address part of the advertisements.

The most amount of traffic is generated by the push method. In this case, the increase in the traffic is in the DT protocol traffic because in the push method, the complete address lists of nodes are included in the protocol messages every time they are advertised. The inclusion of the complete address list information with each node advertisement is particularly inefficient with the DT protocol, because in each heartbeat, it advertises two nodes (i.e., clockwise and counter-clockwise neighbors) to all of its neighbors. The least amount of traffic is generated when no third-party address exchange method is used (i.e., none method) since no substrate address information is being transmitted in the protocol messages.

Note that although in both none and gossiping methods, no substrate address information is being transmitted in DT messages, the DT traffic with the gossiping method

is about 50% higher than with the none method. This is because the gossiping method achieves a much higher percentage of stability which means that, on average, nodes have more neighbors than with the none method. A larger number of neighbors means a larger number of DT message exchanges in each heartbeat. Therefore, although both methods produce DT messages of the same size, the total number of messages and thus the average incoming traffic are higher in the gossiping method.

Compared to the push method, the pull method generates considerably less protocol traffic. The reason is that the pull method only transmit the logical address of advertisers node in node advertisements, whereas the push method transmits the complete address list. In the pull method, the receiver of the advertisement requests the address list only when it is needed regardless of the number of times it has been received in node advertisements, while in the push method the address list is transmitted in each node advertisement. Given that the pull method has achieved good results regarding the connectivity and stability of the DT network, it can be concluded that the pull method offers the best tradeoff overall.

The volume of protocol traffic generated by the push-single method is about the same as the protocol traffic generated by the pull method. This is not surprising because the size of the DT logical address (8 bytes) is close to the size of an a single substrate address in our experiment, i.e., the size of an IP address, a port number, in addition to the size of address realm hash , address type and address length (10 bytes). However, as discussed before, the push-single method only achieves about 80% stability compared to 100% stability achieved by the pull method. This is because the push-single provides nodes with only a single substrate address of the advertised node which may not be sufficient in a multi-substrate environment. Note that the 80% number for push-single depends on the arrangement of the nodes can be different with a different arrangement. As seen in Figures 6.4 and 6.5, the combination of the gossiping method with the push-single method was able to match the performance of the pull method. However, this

performance gain comes at the cost of increased CSA traffic due to gossips.

Finally, as it can be seen in Figure 6.6, the traffic generated by the gossiping method is proportional to the length of the gossiping interval.

6.3 Experiment 2: 2592 nodes and 289 substrates (17x17)

We now repeat the same experiment as before for a larger scale. The number of regions, and the total number of nodes in this experiment is increased by a factor of 4 while number of nodes per region is kept the same. Thus, the outcome will provide insights into the impact of increasing the number of substrates. In this experiments an overlay network with 2592 nodes is constructed. The nodes are equally divided among 324 regions, i.e., 8 nodes per region. Each four neighboring regions form a substrate with 32 nodes. These 289 substrates are arranged in a 17x17 grid. As in the previous experiment, each data point is the average over 3 runs with the same initial values.

Figures 6.7 shows how the stability of the overlay network changes over time with the CSA methods. Comparing this figure with Figure 6.4 confirms that increasing the number of substrates does not have much impact. Figure 6.8 shows how number of leaders decreases over times. The result are similar to Experiment 1.

Table 6.2 and Figure 6.9 include the average CSA and DT traffic received by each node. The difference with the result in Experiment is small confirming that the average traffic of CSA methods and DT protocol is not sensitive to the number of substrates.

Figure 6.10 shows the average number of address lists that nodes have in their address repositories. With the none configuration, each nodes has at most three address lists in its address repository. At each node, the three address lists are for the node itself and for its two configured buddies which have been obtained with a direct address exchange

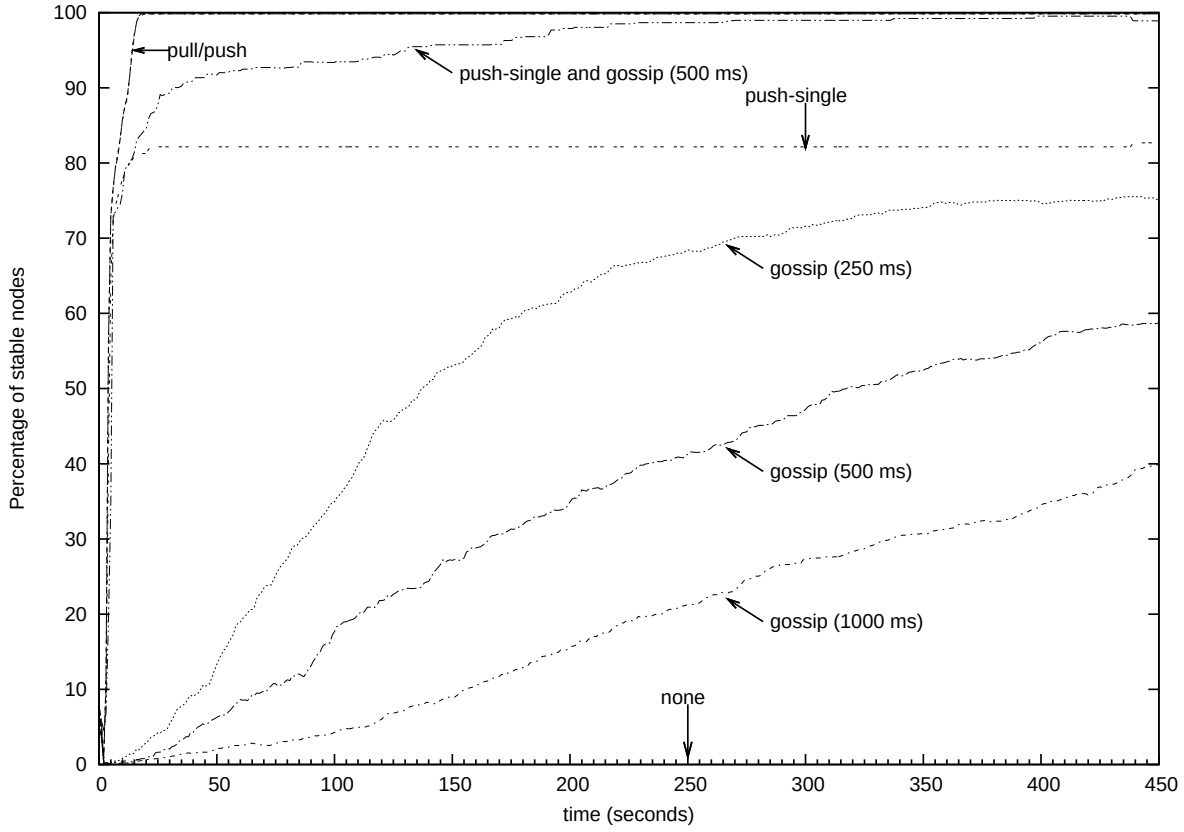


Figure 6.4: Stability of the DT network with 648 nodes in 64 substrates (arranged in a 8x8 grid).

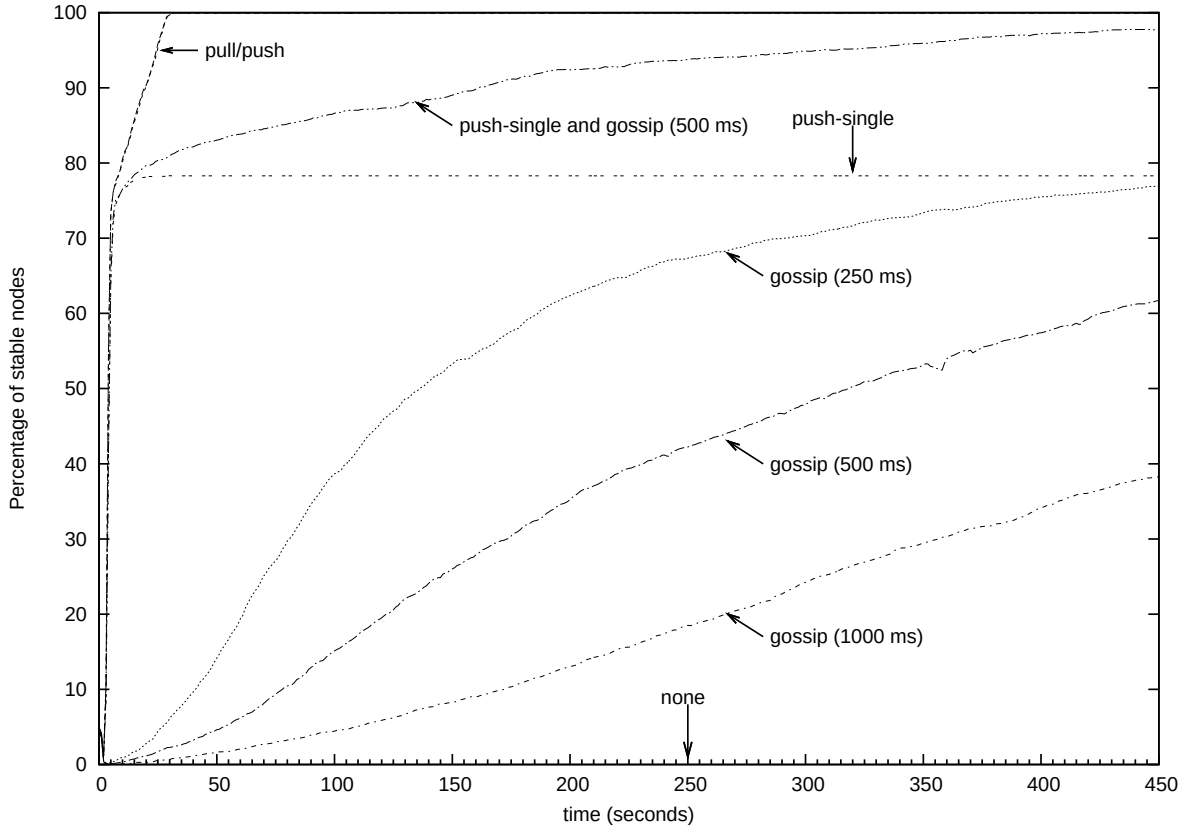
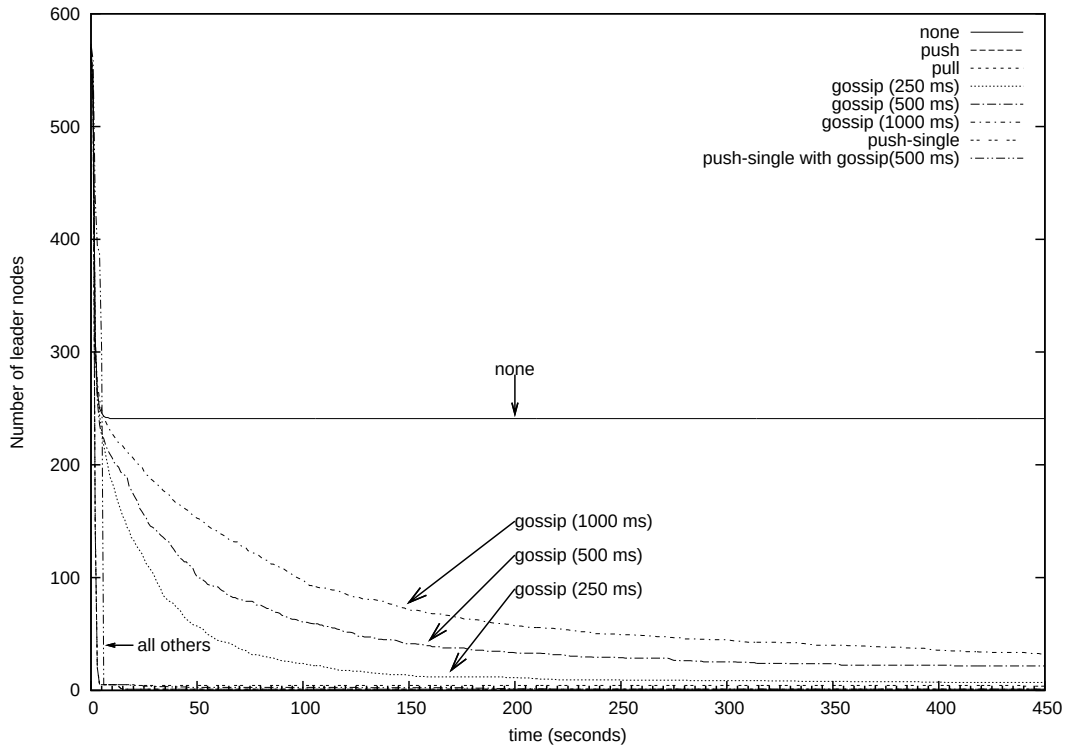
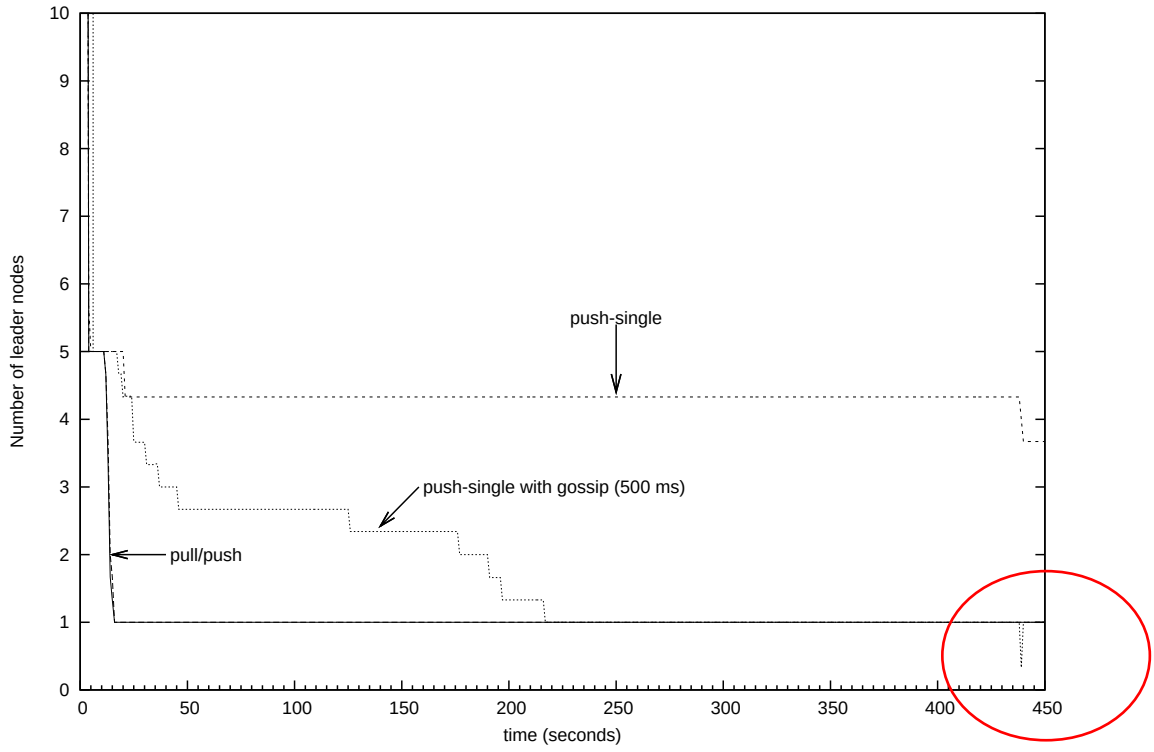


Figure 6.7: Stability of the DT network with 2592 nodes in 289 substrates (arranged in a 17x17 grid).



(a) All methods



(b) Only methods which achieve a high level of connectivity

Figure 6.5: Connectivity of the DT network with 648 node in 64 substrates (arranged in an 8x8 grid): (a) contains all methods (b) contains only those methods which have achieved a high level of connectivity.

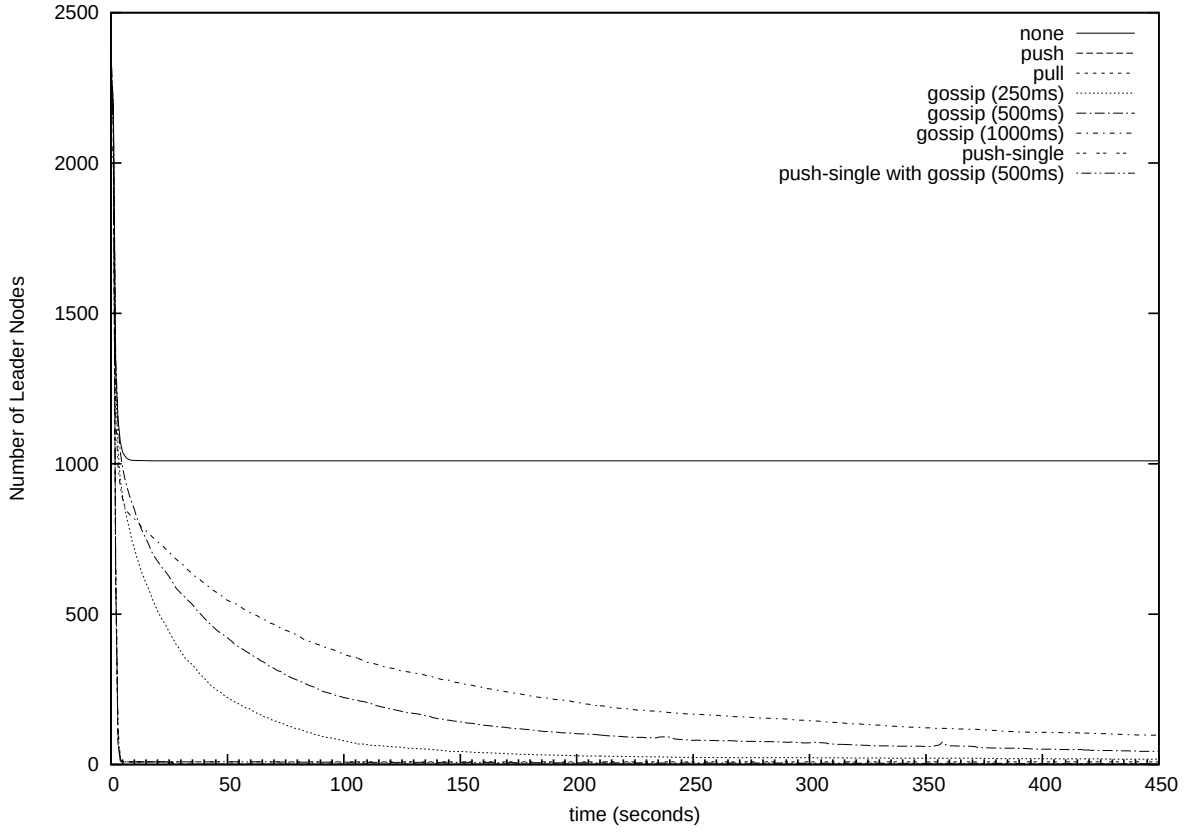


Figure 6.8: Connectivity of the DT network with 2592 nodes in 289 substrates (arranged in a 17x17 grid).

connected overlay network with one leader. The push-single method which exchanges one substrate address decreases the number of leaders to about 5, however, it does not decrease the number of leaders any further. When combined with the gossiping method (push-single & gossiping), it achieves complete connectivity in about 300 seconds. Note that the push and pull method both achieve full connectivity after about 15 seconds.

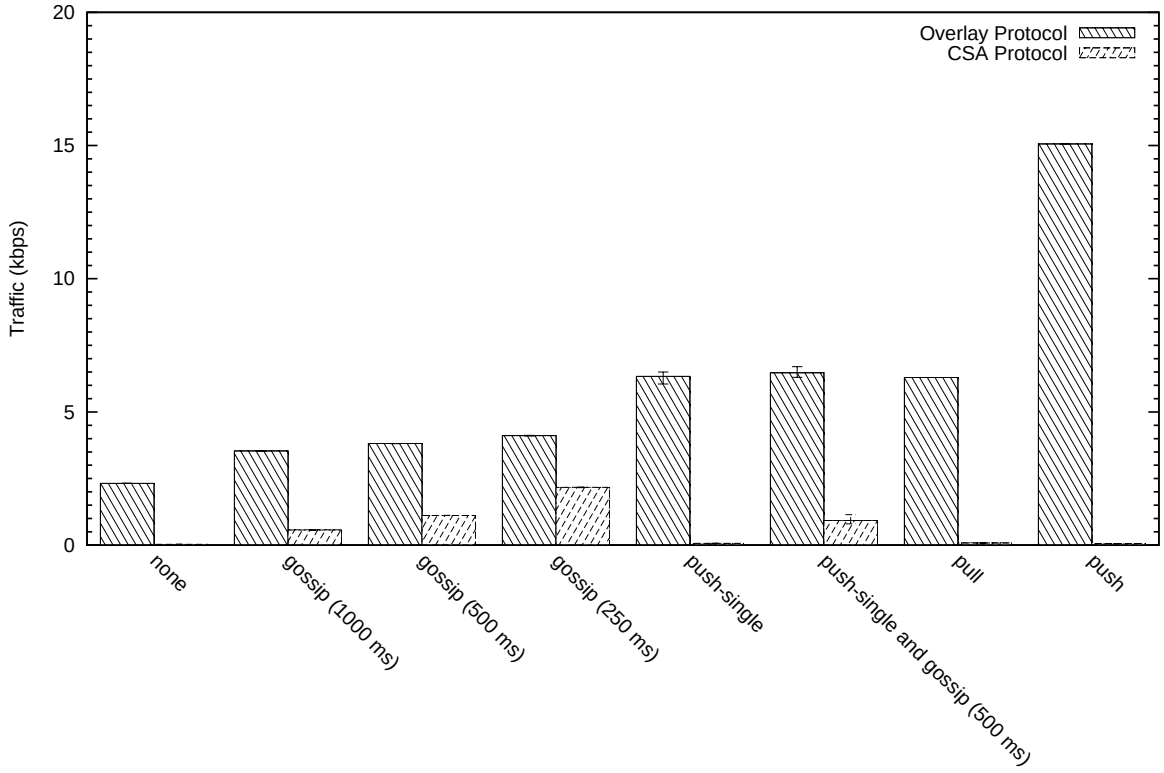


Figure 6.6: Average CSA and DT traffic received by each node of the overlay network with various third-party address exchange methods.

Table 6.1 contains the average CSA and DT protocols traffic that each node has received during the experiment. We measure the received traffic because the node reports of their transmitted traffic may not be the actual transmitted traffic as messages may be dropped at CSA processor or the adapter due to the unavailability of substrate address information.

Figure 6.6 presents the data from Table 6.1 as a graph. The error bar for each bar

Table 6.2: Average and standard deviation of the received CSA and DT traffic for 289 substrates.

Method	CSA (bps)	stddev	DT (bps)	stddev	Total (bps)
none	35.19	56.23	2347.93	3305.09	2383.12
gossip(1000ms)	643.81	836.92	3597.72	2168.30	4241.53
gossip(500ms)	1158.63	1521.63	4344.48	1782.10	5503.11
gossip(250ms)	2311.99	3099.60	4079.49	1413.61	6391.48
push-single	74.67	35.84	6359.99	1863.33	6434.66
push-single & gossip(500ms)	1213.63	1405.86	6643.47	1867.93	7857.1
pull	85.94	39.15	6339.71	1791.46	6425.65
push	59.65	15.83	16173.47	5069.95	16233.12

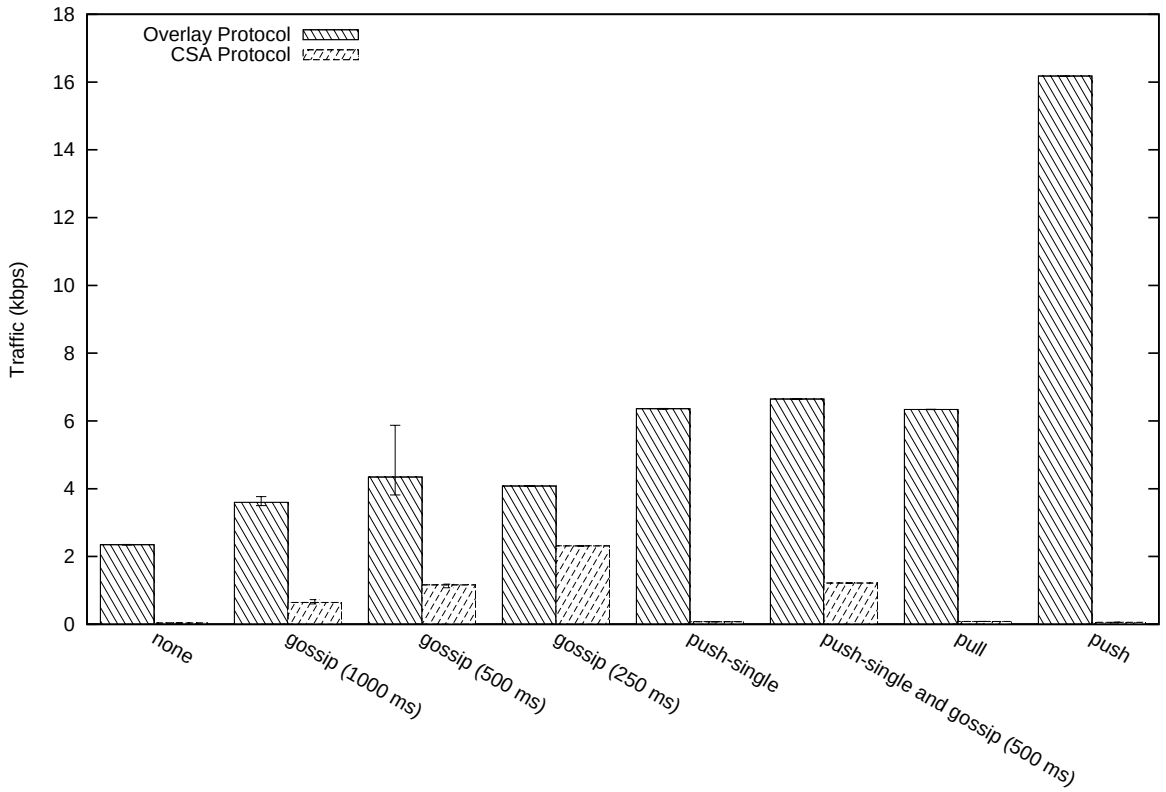


Figure 6.9: Traffic generated by CSA and DT protocols with various methods of third-party address exchange in 289 substrates (arranged in a 17x17 grid).

method. With the push, pull, and push-single methods, each node on average has about the same number of address lists in its address repository, which is about 8 to 9. In addition to the address lists of itself and its two buddies, the address repository contains the address lists for about 5 to 6 neighbors (the average number of neighbors of a node in a Delaunay triangulation graph is 6). Note that this number is achieved at around 90 seconds into the experiment, which is around the time that the first set of unused address lists expires and is removed from the address repository. This confirms that the address list timeout mechanism properly prevents unused advertised address lists to be permanently stored in the address repository.

All three variations of the gossiping method, initially have a linear increase in their number of address lists until they reach a stable number of address lists with a steeper increase for shorter gossiping intervals. The number of address lists around which the method becomes stable depends on the number of address lists which can be in circulation between two consecutive `AddressListTimeoutTimer` events, as this many address list will be updated and thus remain in the address repository. This value itself depends on the gossiping interval and is larger for smaller gossiping intervals.

After reaching stability, the number of address lists still decreases overtime at each `AddressListTimeoutTimer` event (i.e., every 30 seconds), however, these removed address lists are replaced with new ones learned through gossiping process. This creates the visible zig-zag pattern of curves that starts after they reach their peak. The total number of address lists decreases after this peak because, as explained before the rate by which a node receives new address lists is decreasing over time. This can be improved by improving the gossiping scheme.

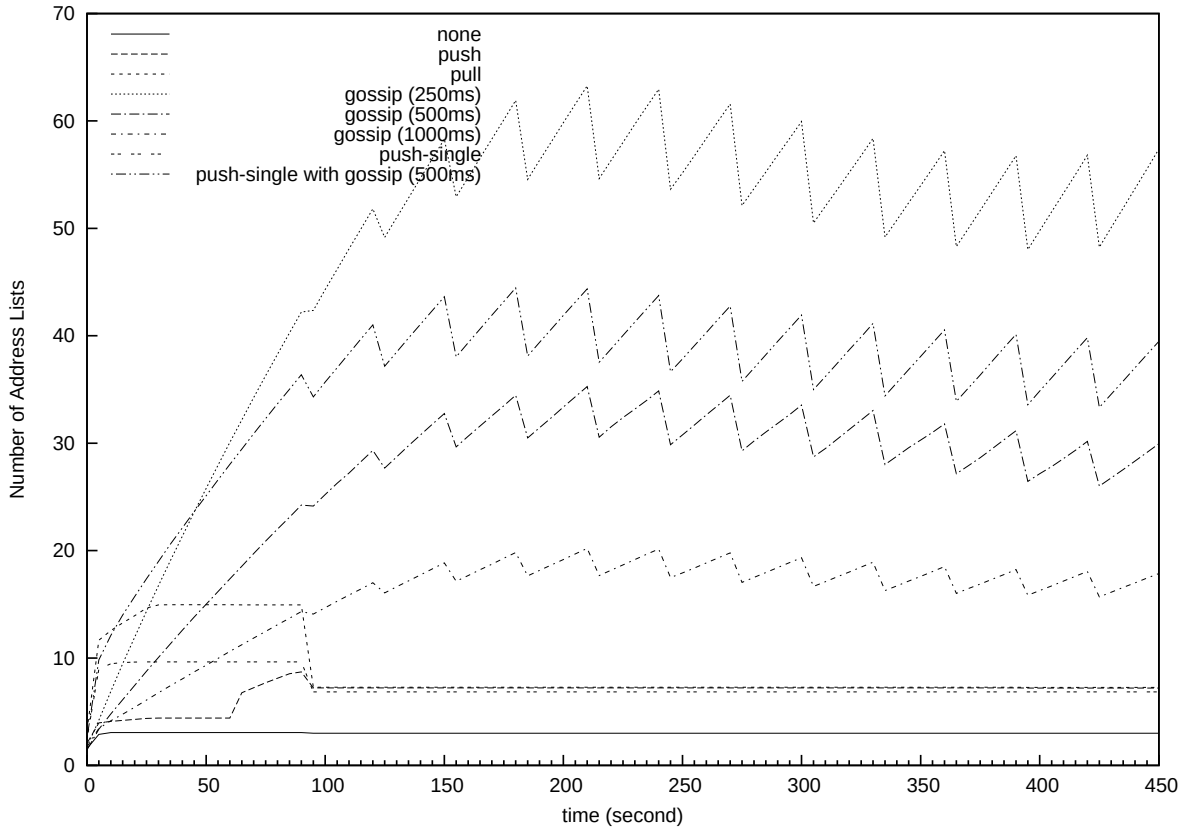


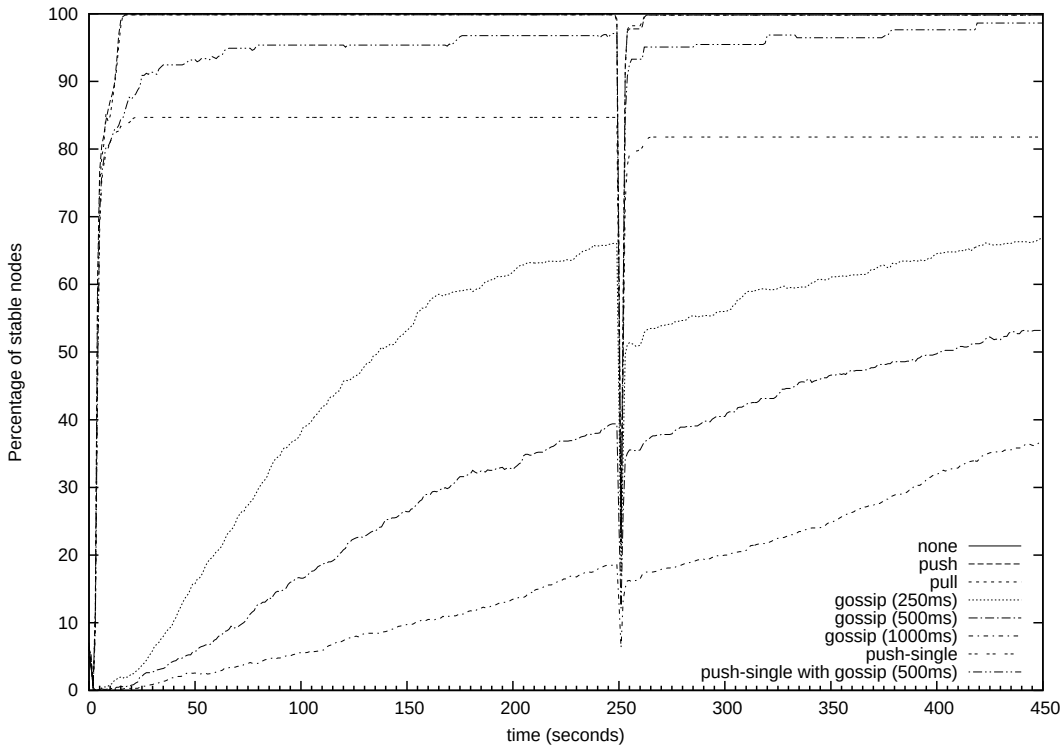
Figure 6.10: Number of address lists with various third-party address exchange methods in a 17x17 grid.

6.4 Experiment 3: Performance of CSA in a Dynamic Environment under Churn

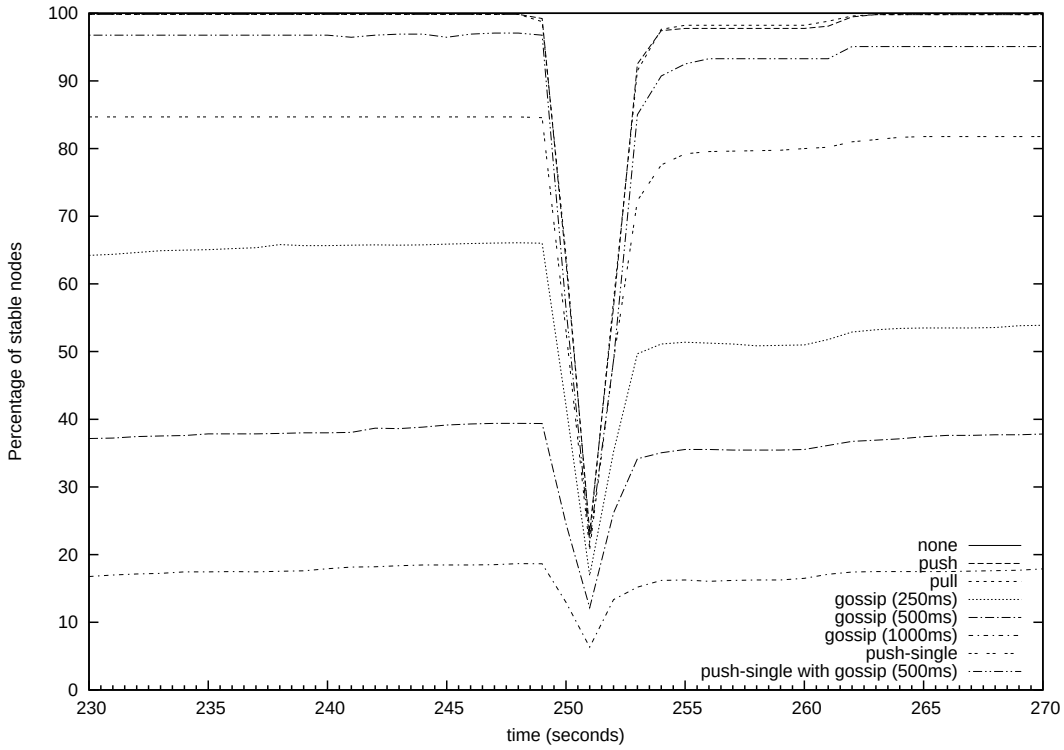
In this experiment we examine how each third-party address exchange method performs in a situation in which many nodes depart simultaneously from the network. The setup for this experiment is as in Experiment 1, with the difference that half-way through the experiment (at time 250 seconds) 25% of the nodes in each region leave the network. We are interested in comparing how each method behaves after the departures of the nodes to gain insight into the robustness of the CSA methods after churn.

Figure 6.11(a) shows the stability of the network with different CSA methods over running time of the experiment. Figure 6.11(b) shows the same data in the time interval between 230 seconds and 270 seconds which is 20 seconds before and after the departure event.

Until the departure event the outcomes are very similar to those seen in Figure 6.4. When the departure event happens, overlay nodes whose neighbors have left the overlay network, have to establish neighborhood relation with nodes which are being advertised to them for the first time. After the departure event, the push and pull methods recover quickly as they are able to quickly obtain the address lists for such newly advertised nodes. The gossiping method, requires a longer time to regain the same level of stability that it had before the departure event. This is due to the fact that it takes a longer time for the gossiping method to obtain the address list of newly advertised nodes compared to the push and pull methods.



(a) Result of the entire experiment



(b) Detailed snapshot in the time interval between 230 seconds and 270 seconds.

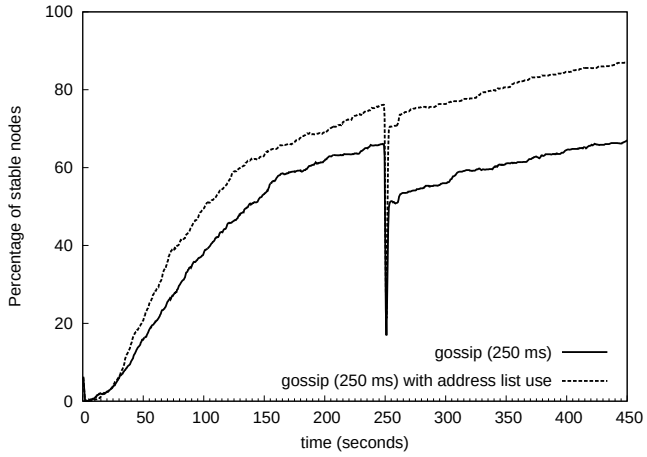
Figure 6.11: Stability of DT network in a situation with churn.

6.5 Experiment 4: Gossiping as a Source of Out-of-Band Address Information

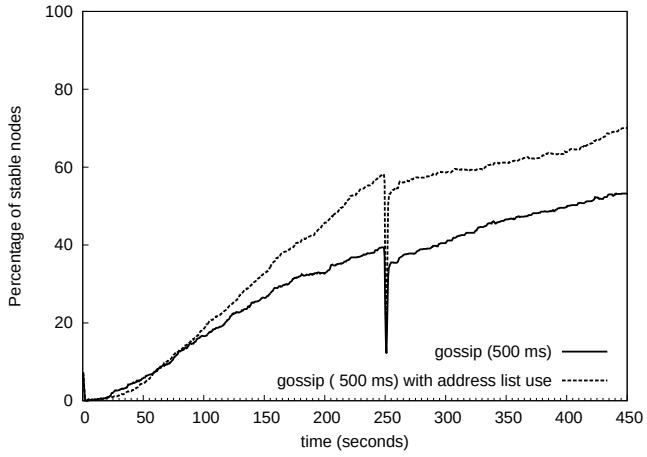
As discussed in Section 4.2.2, as part of the gossiping process, each node receives address lists that are not directly useful for the node itself but are kept for the sole purpose of being gossiped to other nodes. This address lists exchanged by the gossiping method can be a valuable out-of-band source of information for the rendezvous mechanism of an overlay protocol. To demonstrate how this information can be used, we have created a modified version of DT buddylist protocol which takes advantage of the address lists acquired by gossiping method in its rendezvous process in addition to its statically configured buddy addresses.

The DT buddylist protocol sends an `IAmHere` to one randomly chosen buddy among all its buddies (two in this case). The modified DT protocol sends an `IAmHere` message to five randomly chosen gossiped address lists from its address repository in addition to the buddy.

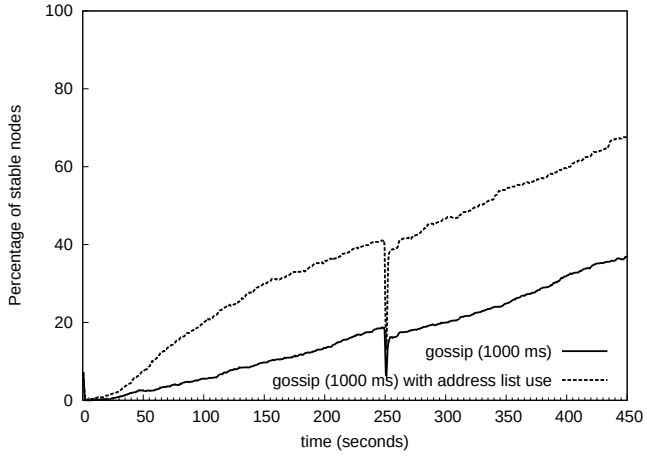
Figure 6.12 shows how the use of gossiped address lists in the rendezvous process improves the speed by which the DT protocol becomes stable in a setup as in Experiment 3. In Figure 6.12(a) shows the improvement when gossiping (250 ms) is used and Figures 6.12(b) and 6.12(c) respectively show the improvement with gossiping (500 ms) and gossiping (1000 ms) methods. As it can be seen in these figures, using the address lists obtained through gossiping, the overlay nodes improve their rendezvous process. As a result, the overlay networks stabilized in a shorter amount of time after the start of the experiment and after the departure event. This improvement is more apparent anywhere that the original stability level is low e.g., the gossiping (1000 ms) method.



(a) Gossiping (250 ms)



(b) Gossiping (500 ms)



(c) Gossiping (1000 ms)

Figure 6.12: Improvement in the stability of the DT network with the gossiping method after modification to the DT protocol, where gossiped address lists are used in the rendezvous process.

6.6 Experiment 5: CSA Performance in Broadcast Substrates

In this experiment, we try to gain insight in the effectiveness of the broadcast address exchange method for overlay networks that use broadcast substrates. For this experiment we use the DT broadcast protocol which is a version of the DT protocol that uses broadcast messages for rendezvous. In the DT broadcast protocol, the leader of the network broadcasts a beacon every *BeaconTime* when a node receives this message it will contact the leader if does not have a better neighbor. In the steady state there is only one leader and thus only one node broadcasts DT messages.

In this experiment, only two substrate networks are used, which consist of two Ethernet LANs with different rates (100 Mbps, 1 Gbps). We use 8 hosts for this experiment. Overlay nodes are equally distributed across the hosts.

We use an overlay network with 1024 overlay nodes for this experiment. Each overlay socket is configured with two interfaces: a UDP multicast interface which is connected to the 100 Mbps LAN which is used for rendezvous messages, and a UDP unicast interface which is connected to the 1 Gbps LAN which is used for all other messages. We use the same parameters for DT and CSA as in the previous experiment, except that DT now has a *BeaconTime* parameter which is set to 1000 ms.

Two variations of the CSA broadcast method are tested in this experiment. The first variation, referred to as *piggyback broadcast*, always piggybacks the complete address list of the sending node on any outgoing broadcast message. In the other variation, referred to as *periodic broadcast*, each node piggybacks its address list on the first outgoing broadcast message after which it starts to periodically broadcast its address list on its broadcast interface. The push method is used for third-party address exchanges.

Figure 6.13 shows the stability and connectivity of the DT Broadcast network over time. With both variations of the CSA broadcast method, the network becomes stable

and connected in about 10 seconds. Since with both variations the first broadcast of each node always includes the sender's address list, we do not expect a big difference in the outcome. The small difference between these method is due to slight differences that exist between the times that nodes send their first broadcast. In DT broadcast, each nodes waits a random amount of time between 0 and *BeaconTime* before broadcasting its first beacon which causes the slight difference in the outcomes.

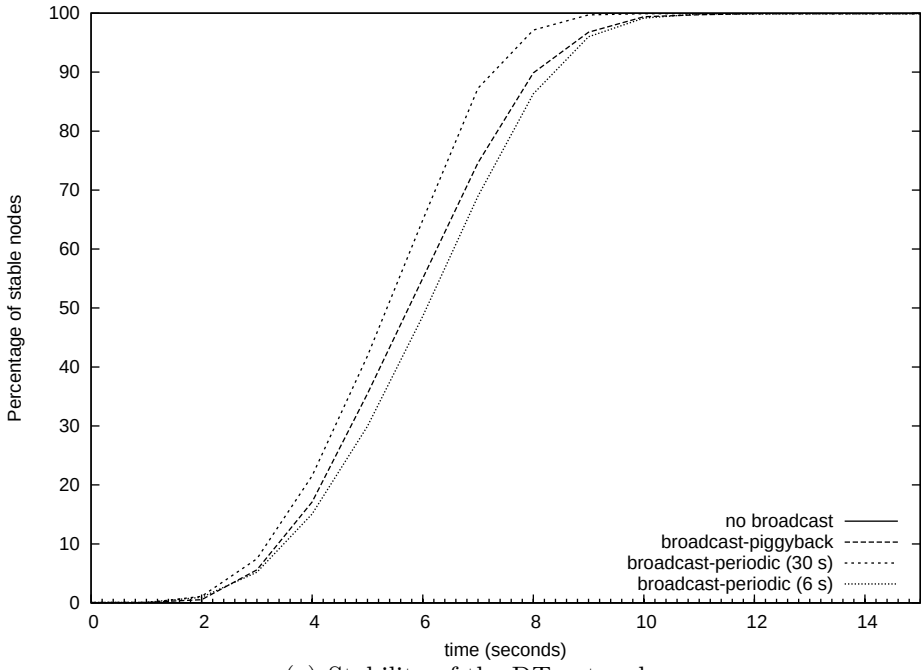
There is a significant difference between the CSA broadcast methods with respect to the amount of traffic generated by them. Table 6.3 and Figure 6.14 show the average received protocol traffic generated by each method. The most efficient method in terms of the protocol traffic is the piggyback broadcast method. The most traffic is generated by periodic broadcast with a broadcast period of 6 seconds. The traffic generated by this method is 50 % more that the traffic generated by the DT protocol. A better choice for the broadcast period is to set it to a fraction of the of the address list timeout value. This ensures that address lists will not be timed out in address repositories while limiting the broadcast traffic at the same time. The 30 seconds broadcast period is half the address list time out value (60 seconds). The periodic broadcast with broadcast period of 30 seconds significantly reduces the protocol traffic.

Although the broadcast period was increased by a factor a 5, the reduction in the traffic is a factor of 6. This is because the initial number of leaders when broadcast period of 30 seconds is used, is smaller which means that a smaller number of nodes send broadcast messages.

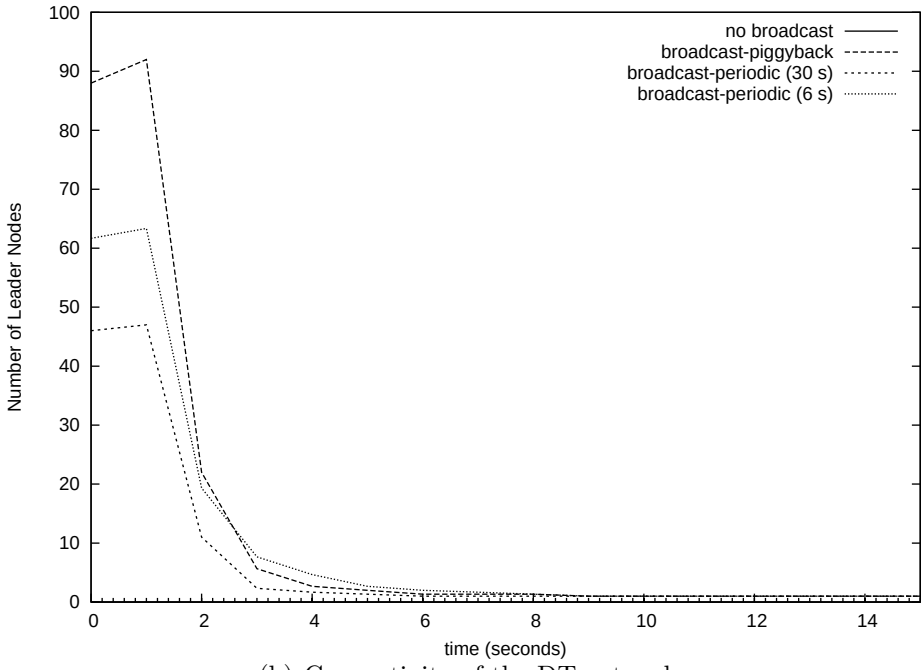
We note that the results are specific to the DT broadcast protocol. Periodic broadcast may be more efficient with different protocols. This is because, in DT broadcast, all nodes except the leader node send only few broadcast messages when they are joining the network, but otherwise do not broadcast messages.

Table 6.3: Average and standard deviation of the received CSA and DT traffic with broadcast method.

Method	CSA (bps)	stddev	DT (bps)	stddev	Total (bps)
broadcast-periodic (6s)	13221.76	85.35	8539.31	1840.17	21761.07
broadcast-periodic (30s)	1978.22	97.85	8517.20	1847.01	10495.42
broadcast-piggyback	942.27	162.60	8644.23	1845.37	9586.50



(a) Stability of the DT network



(b) Connectivity of the DT network

Figure 6.13: Stability (a) and connectivity (b) in a DT Broadcast network with 1024 nodes.

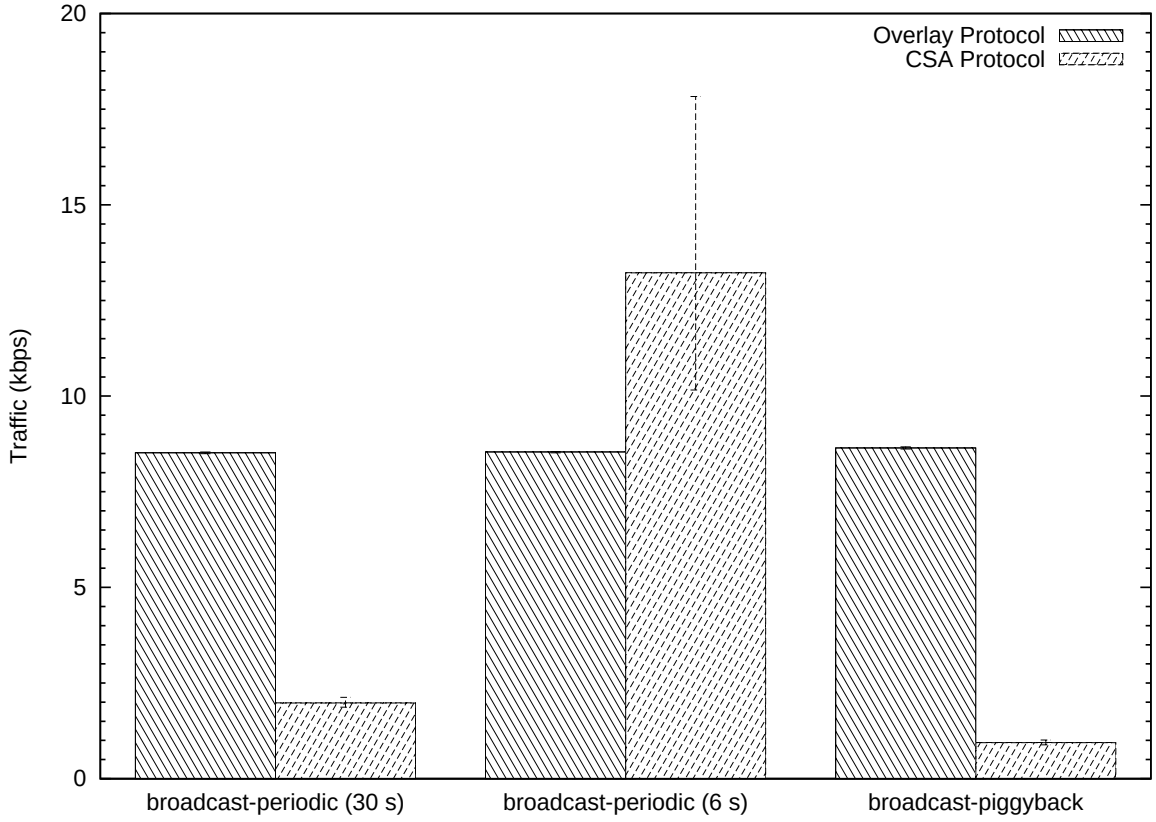


Figure 6.14: Average received traffic with different broadcast variations in a DT Broadcast network with 1024 nodes.

6.7 Summary

In our experiments, the pull method appeared as the most efficient third-party address exchange method. The push method yields a good performance in terms of disseminating address lists but incurs more overhead in terms of protocol traffic. A combination of the push-single method with the gossiping method improves the performance of the gossiping method while incurring a modest protocol traffic.

We showed that all third-party address exchange methods are able to recover quickly after a significant disruption in the network. Furthermore, we showed that it is possible to improve the performance of an overlay protocol by taking advantage of the opportunity of gossiped address list information.

Finally, we analyzed and discussed variations of the broadcast method for direct address exchange.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

In this thesis, we presented cross-substrate advertisement (CSA), a mechanism that improves the ability of overlay networks to be effective in an environment of heterogeneous substrate networks. CSA assists the overlay networks to deal with one-to-many address bindings that are inherent to environments with multiple substrate networks.

We presented two categories of address exchanges for self-organizing overlay networks referred to as direct address exchange and third-party address exchange. For direct exchange we proposed two methods: the request method for exchanging and updating address bindings between communicating pairs of overlay nodes in unicast substrates, and the broadcast method for direct address exchanges in broadcast substrate networks. For third-party address exchange, we proposed push, pull, and gossiping methods. Our evaluation shows that the pull method is superior in achieving desirable network properties without incurring high overhead. An important contribution of our work is that we identified crucial issues that must be considered when designing a self-organizing overlay protocol for an environment with heterogeneous substrate networks. And finally, our solution allow existing HyperCast overlay protocols to be able to operate in a multi-

substrate environment with minimal modifications.

7.2 Future Work

This thesis offers opportunities to be extended and continued in the following directions.

1. **Investigating Partial Address Exchange:** With the exception of the push-single method, all third-party address exchange methods only exchange complete address lists. It is possible to design more sophisticated strategies to exchange a subset of address lists that are tailored to the sender and receiver of a node advertisement. Such exchanges of partial address lists can become a factor in situations where nodes have many interfaces.

2. **More Flexible Gossiping Schemes:** We use a simple gossiping scheme that does not take into account the topology of the overlay network. Our gossiping scheme could be modified to suit a particular overlay topology. It may be feasible to design a versatile gossiping scheme whose address list dissemination performance can be tuned for particular overlay protocols.

3. **Unstructured Peer-to-Peer Overlay Protocol:** In our evaluation we used CSA mechanism in structured overlay networks (Delaunay triangulation). Since, in a multi-substrate environment, nodes are restricted to choose their neighbors from substrate networks which they have access to, it is sometimes not feasible to construct a complete structured overlay topology. Unstructured overlay networks are better suited to deal with interconnecting heterogeneous networks. The effectiveness CSA with an unstructured overlay network remains to be investigated.

4. **Automatic Address Realm Discovery:** Currently in our design, address realms are statically configured. Such static configuration is not desirable for self-organizing

peer-to-peer networks. It may be possible to design a method for automatic selection or detection of an address realm for each substrate. The authors of [28] have proposed one possible solution which is based on a gossiping approach. It remains to be seen what other approaches can be used for this purpose and how they can be used with CSA.

Bibliography

[1] Akamai. http://www.akamai.com.

[2] David G. Andersen, Hari Balakrishnan, Frans M. Kaashoek, and Robert Morris. Resilient overlay networks. In *Proc. of the ACM Symposium on Operating Systems Principles (SOSP 2001)*, pages 131–145, 2001.

[3] Thomas Anderson, Larry Peterson, Scott Shenker, and Jonathan Turner. Overcoming the Internet impasse through virtualization. *Computer*, 38(4):34–41, 2005.

[4] Börje Ohlman Bengt Ahlgren, Lars Eggert and Andreas Schieder. Ambient networks: bridging heterogeneous network domains. In *Proc. of the 16th Annual IEEE International Symposium on Personal Indoor and Mobile Radio Communications (PIMRC 2005)*, Berlin, September 2005.

[5] Roland Bless, Christian Hübsch, Sebastian Mies, and Oliver P. Waldhorst. The underlay abstraction in the spontaneous virtual networks (SpoVNet) architecture. In *Proc. of the Next Generation Internet Networks (NGI 2008)*, pages 115–122, April 2008.

[6] Randy Bush and David Meyer. Some Internet architectural guidelines and philosophy. RFC 3439, Internet Engineering Task Force, December 2002.

[7] Isidro Castineyra, Noel Chiappa, and Martha Steenstrup. The Nimrod Routing Architecture. RFC 1992, Internet Engineering Task Force, August 1996.

[8] Yang-Hua Chu, Sanjay G. Rao, Srinivasan Seshan, and Hui Zhang. A case for end system multicast. In *Proc. of the ACM Sigmetrics*, pages 1–12, 2002.

[9] Dave Clark, Bill Lehr, Steve Bauer, Peyman Faratin, Rahul Sami, and John Wroclawski. Overlay networks and the future of the Internet. *COMMUNICATIONS & STRATEGIES*, 63(3), 2006.

[10] David Clark. The design philosophy of the DARPA Internet protocols. *SIGCOMM Comput. Commun. Rev.*, 18(4):106–114, 1988.

[11] David Clark, Robert Braden, Aaron Falk, and Venkata Pingali. FARA: reorganizing the addressing architecture. In *Proc. of the ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA 2003)*, pages 313–321, 2003.

[12] Jon Crowcroft, Steven Hand, Richard Mortier, Timothy Roscoe, and Andrew Warfield. Plutarch: an argument for network pluralism. In *Proc. of the ACM SIGCOMM Workshop on Future Directions in Network Architecture (FDNA 2003)*, pages 258–266, 2003.

[13] Aaron Falk David D. Clark, Scott Shenker. GENI research plan. Technical report, Research Coordination Working Group, April 2007.

[14] Kevin Fall. A delay-tolerant network architecture for challenged Internets. In *Proc. of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer communications*, pages 27–34, New York, 2003.

[15] Anja Feldmann. Internet clean-slate design: what and why? *SIGCOMM Comput. Commun. Rev.*, 37(3):59–64, 2007.

[16] Paul Francis. Pip near-term architecture. RFC 1621, Internet Engineering Task Force, May 1994.

[17] Paul Francis and Ramakrishna Gummadi. IPNL: A NAT-extended Internet architecture. In *Proc. of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 69–80, 2001.

[18] Mark Gritter and David R. Cheriton. An architecture for content routing support in the Internet. In *Proc. of the 3rd USENIX Symposium on Internet Technologies and Systems (USITS 2001)*, 2001.

[19] Sandra M. Hedetniemi, Stephen T. Hedetniem, and Arthur L. Liestman. A survey of gossiping and broadcasting in communication networks. *Networks*, 18(4):319–349, 1988.

[20] Andreas Jonsson, Mats Folke, and Bengt Ahlgren. The split naming/forwarding network architecture. In *Proc. of the First Swedish National Computer Networking Workshop (SNCNW 2003)*, Arlandastad, 2003.

[21] Richard M. Karp, Christian Schindelhauer, Scott Shenker, and Berthold Vöcking Schindelhauer. Randomized rumor spreading. In *Proc. of the 41st IEEE Annual Symposium on Foundations of Computer Science (FOCS 2000)*, pages 565–574, 2000.

[22] David Kempe and Jon M. Kleinberg. Protocols and impossibility results for gossip-based communication mechanisms. In *Proc. of the 43rd IEEE Annual Symposium on Foundations of Computer Science (FOCS 2002)*, pages 471–480, 2002.

[23] Jörg Liebeherr. Monitor and control system for HyperCast. http://www.comm.utoronto.ca/hypercast/design/MonitorAndControlv20.pdf.

[24] Jörg Liebeherr, Michael Nahas, and Weisheng Si. Application-layer multicasting with Delaunay triangulation overlays. *IEEE Journal on Selected Areas in Communications*, 20(8):1472–1488, October 2002.

[25] Jörg Liebeherr, Jianping Wang, and Guimin Zhang. Programming overlay networks with overlay sockets. In *Proc. of the 5th COST 264 Workshop on Networked Group Communications (NGC 2003), LNCS 2816*, pages 242–253, 2003.

[26] John W. Lockwood, Nick McKeown, Greg Watson, Glen Gibb, Paul Hartke, Jad Naous, Ramanan Raghuraman, and Jianying Luo. NetFPGA–an open platform for Gigabit-rate network switching and routing. In *Proc. of the IEEE International Conference on Microelectronic Systems Education (MSE 2007)*, pages 160–161, 2007.

[27] Nick McKeown and Bernd Girod. Clean-slate design for the Internet, April 2006. White paper: http://cleanslate.stanford.edu.

[28] Sebastian Mies, Oliver Waldhorst, and Hans Wippel. Towards end-to-end connectivity for overlays across heterogeneous networks. In *Proc. of the International Workshop on the Network of the Future (Future-Net 2009), co-located with IEEE International Conference on Communications (ICC 2009)*, Dresden, June 2009.

[29] Pablo Molinero-Fernández, Nick McKeown, and Hui Zhang. Is IP going to take over the world (of communications)? *SIGCOMM Comput. Commun. Rev.*, 33(1):113–118, 2003.

[30] Robert Moskowitz and Pekka Nikander. Host Identity Protocol (HIP) architecture. RFC 4423, Internet Engineering Task Force, May 2006.

[31] Mike O'Dell. GSE-an alternative addressing architecture for IPv6. RFC draft-ietf-ipngwg-gseaddr-00, Internet Engineering Task Force, February 1997.

[32] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Schenker. A scalable content-addressable network. In *Proc. of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 161–172, 2001.

[33] Antony I. T. Rowstron and Peter Druschel. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *Proc. of the IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, pages 329–350, 2001.

[34] Ignacio Solis and Katia Obraczka. FLIP: A flexible interconnection protocol for heterogeneous internetworking. *Mobile Networks and Applications*, 9(4):347–361, 2004.

[35] Ion Stoica, Robert Morris, David Karger, Frans M. Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for Internet applications. In *Proc. of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pages 149 – 160, San Diego, August 2001.

[36] Lakshminarayanan Subramanian, Ion Stoica, Hari Balakrishnan, and Randy Katz. OverQoS: An overlay based architecture for enhancing Internet QoS. In *1st USENIX Symposium on Networked Systems Design and Implementation (NSDI 2004)*, San Francisco, March 2004.

[37] Zoltán Turányi, András Valkó, and Andrew T. Campbell. 4+4: an architecture for evolving the Internet address space back toward transparency. *SIGCOMM Comput. Commun. Rev.*, 33(5):43–54, 2003.

[38] Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar. An integrated experimental environment for distributed systems and networks. *SIGOPS Oper. Syst. Rev.*, 36(SI):255–270, 2002.