

A Scalable Architecture for Broadcast Federation

Mukund Seshadri

University of California at Berkeley

Email: mukunds@cs.berkeley.edu

Abstract

Several protocols have been proposed to enable efficient multi-point communication (multicast or broadcast) both at the IP layer and at the application or overlay layer. However, no single protocol is universally deployed. Instead, we expect the existence of islands of non-interoperable broadcast connectivity. In the absence of a scheme to share broadcast connectivity across these “islands”, clients in an island cannot receive broadcast information from sources in a different island. To address this situation, we propose an architecture that enables the composition of different multicast- or broadcast-capable networks, each with their own internally deployed multicast or broadcast protocols, to provide an end-to-end broadcast service. We call this architecture the *Broadcast Federation*. This architecture is based on a network of overlay-level peering links between Broadcast Gateways. In order to achieve high throughput- and session-scalability, each Broadcast Gateway is designed as a cluster of workstations. We have implemented this architecture and tested it on a university cluster of workstations. This document describes this architecture and the results pertaining to this implementation.

1 Introduction

One-to-many and many-to-many broadcasting ¹ applications such as live Internet streaming, multimedia conferencing, and online software distribution are gaining in popularity. These applications are experiencing increasingly large demand driven in part by a growing user

¹In this document, we use the term “broadcast” to mean one-to-many or many-to-many transmission, instead of the traditional term “multicast”, which is often construed to mean the network-layer IP multicast protocol [10].

population with broadband Internet connections. As an example, an online Madonna concert recorded nine million clients [9]. Applications that have such a large client population need efficient one-to-many (or many-to-many) communication primitives within the network.

For over a decade, researchers have spent considerable effort on the design of protocols to support such efficient broadcasting. The IP Multicast service [10] was proposed as an extension to the Internet architecture for efficient multi-point packet delivery at the network level. However, as we will see in Section 2, there were several problems with IP Multicast, like address space scarcity and lack of access control.

To address some of these shortcomings, researchers and commercial vendors have developed other network layer protocols such as Source-specific Multicast (SSM) [4] as well as a variety of application-layer overlay protocols such as End-system Multicast [17], Scattercast [7], and Overcast [22]. While many of these protocols (and IP Multicast) can probably be deployed at a small scale or in a small number of administrative domains, we do not expect that any single protocol will be deployed globally across the entire Internet. This is due to a variety of reasons that we shall detail in Section 2. Briefly, these reasons include technical shortcomings in the protocols or their implementations, limitations in scalability, the fact that some of these protocols have been designed to support specific applications, concerns about access control, and the failure to satisfactorily address a range of business model concerns.

Consequently, we are gravitating toward a situation where a number of diverse and non-interoperable protocols co-exist in the Internet. We envision an Internet landscape fragmented into potentially overlapping clouds of broadcast connectivity with no interoperation across these clouds.

To address this situation, we propose an architecture called *Broadcast Federation* for composing the broadcast connectivity offered by different broadcast networks, each of which implements its own independent broadcast protocol. Our architecture interconnects individual broadcast-capable networks, such as an ISP's IP multicast network or a Content Distribution Network spanning multiple ISPs, into a federation of loosely-coupled broadcast systems. The architecture is depicted in Figure 1. At its core is the *Broadcast Gateway (BG)*, which is the point of peering between different *Broadcast Networks (BNs)*. Within a single BN, the archi-

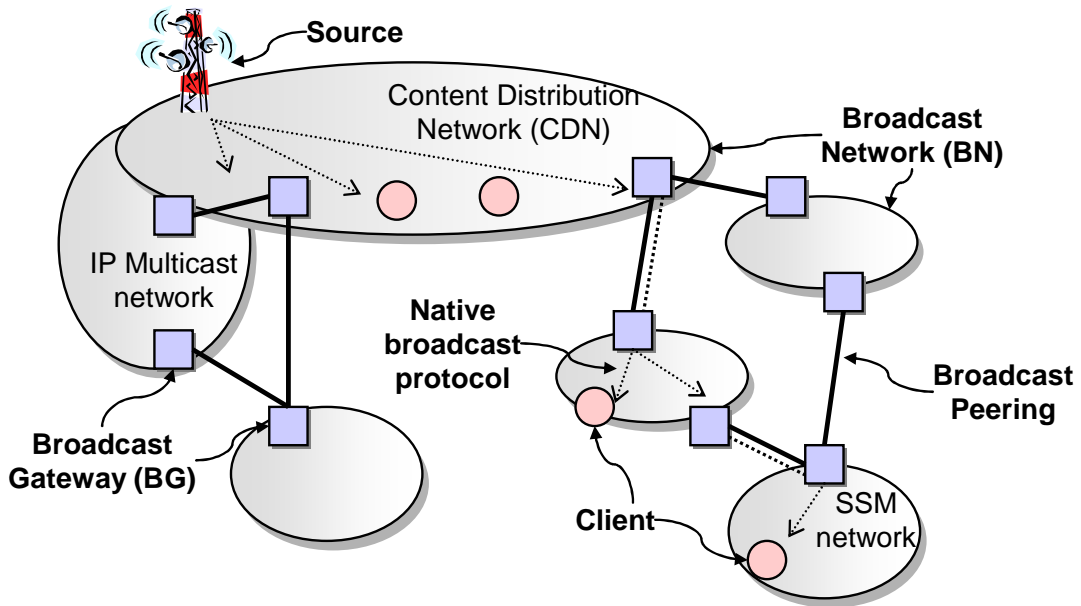


Figure 1: The Broadcast Federation Architecture. *Broadcast Gateways* are explicit application-layer peering points that interconnect a diverse collection of *Broadcast Networks* and translate between the inter-network protocols, and the individual native intra-network protocols.

ecture uses the native broadcast protocols of that network, while the BGs act as translation and forwarding points across networks. Peering relationships between broadcast gateways are administratively configured. The BGs construct an overlay internetwork across the federation by means of a suite of routing, tree-building, and data-forwarding protocols. These protocols apply the principles of network-layer routing protocols such as BGP [30] and PIM [11] across the federation.

To interoperate across a range of native broadcast protocols, each broadcast gateway has a customizable component called the *NativeNet* (*Native network interface*) component that isolates the implementation of the native broadcast capability for that BN. This represents the interface between the federation protocols and the specific protocols used within its own native broadcast network (as we further discuss in Section 5.4). Each broadcast gateway implements a custom version of this component.

At a conceptual level, our architecture is similar to BGMP [23], which provides a solution for inter-domain IP multicast. However, BGMP is a pure IP-layer solution that relies on extensions to BGP to disseminate multicast routing information and uses the IP Multicast addressing scheme ². On the other hand, our architecture is designed to serve both IP and application-level broadcast protocols, and does not depend on the currently deployed BGP framework, or any other IP-level protocols or addressing schemes. Instead we choose an overlay network approach for disseminating reachability information, which allows us to integrate network-layer and application-layer protocols into a seamless global internetwork. This overlay approach also has a well known benefit: the ease of installation and incremental deployment.

One of the primary bottlenecks in the design of such a scalable Broadcast Federation framework is the Broadcast Gateway. All communication across BNs relies on broadcast gateways. Hence it is imperative that this component be capable of scaling up with offered load - while packet processing is not a high-latency task in general, it is likely that the BG will have to handle a very large number of packets per second. To design a scalable Broadcast Gateway, we leverage the observation that workloads at individual BGs are naturally parallelizable: each broadcast stream flowing through a BG is essentially independent of the rest. Such “embarrassingly parallel” workloads are amenable for deployment on clusters of workstations, with minimal management of distributed state, while benefiting from the inherent scaling properties of clusters [15, 8]. First, using commodity workstations as the unit of scaling within a clustered BG allows us to incrementally grow the capacity of the BG (in terms of throughput and number of sessions) over time. Second, the inherent redundancy of clusters can be used to mask transient failures within a BG. Finally, while clusters of commodity workstations require suffer from large space and power requirements, they are easier to deploy and customize than a solution based on special-purpose hardware.

Therefore, in this document, we propose an architecture for the Broadcast Gateway that relies on clusters of workstations. Each BG is composed of a number of individual workstations interconnected by a high-speed network. This “expanded” view of the architecture

²Scarcity of the IP Multicast address space is a well-known problem.

is depicted in Figure 2. Each BG has a single control node that is used for communicating with other BGs across the Federation and for coordinating actions within the BG. In addition to the control node, each BG has a number of data nodes, which handle the forwarding of data packets across broadcast sessions. The capacity of the BG to handle broadcast traffic can be grown “simply” by adding more data nodes to the BG cluster. We have implemented this design of the Broadcast Gateway, and tested the same on a university cluster of workstations.

To summarize our problem: we expect the existence of several diverse broadcast protocols deployed in separate administrative domains or networks in the Internet. In order to enable interoperability between these networks, we require an architecture that allows these networks to obtain reachability information about each other and transmit broadcast data along the most efficient paths (or trees) between sources and clients in different networks. We design such an architecture and identify a potential scalability bottleneck, namely the Broadcast Gateway. We propose a clustered design for the Broadcast Gateway, implement this design, and present results from this implementation.

The rest of this document is organized as follows. In Section 2, we present background and related work. In Section 3, we describe the components and service model of our architecture. Section 4 gives an overview of the design of the BG, and of the interactions between BGs. Section 5 proceeds to describe the details of the protocols used within a single BG cluster and between several BGs. Section 6 describes our implementation and experimental setup, and presents results obtained from the experiments. Finally, we conclude and outline future work in section 7.

2 Background

The Internet community has tried to solve the problem of providing an efficient multi-point data delivery framework for a long time. Deering et al. proposed IP multicast [10] in 1990 as an extension to IP that could support many-to-many packet delivery within routers. A packet sent to a “multicast group” would reach the set of end-hosts that desire to receive that packet, that is, the members of the multicast group. In spite of over a decade of research, IP multicast has failed to reach widespread deployment Internet. In [12], the authors describe many of the

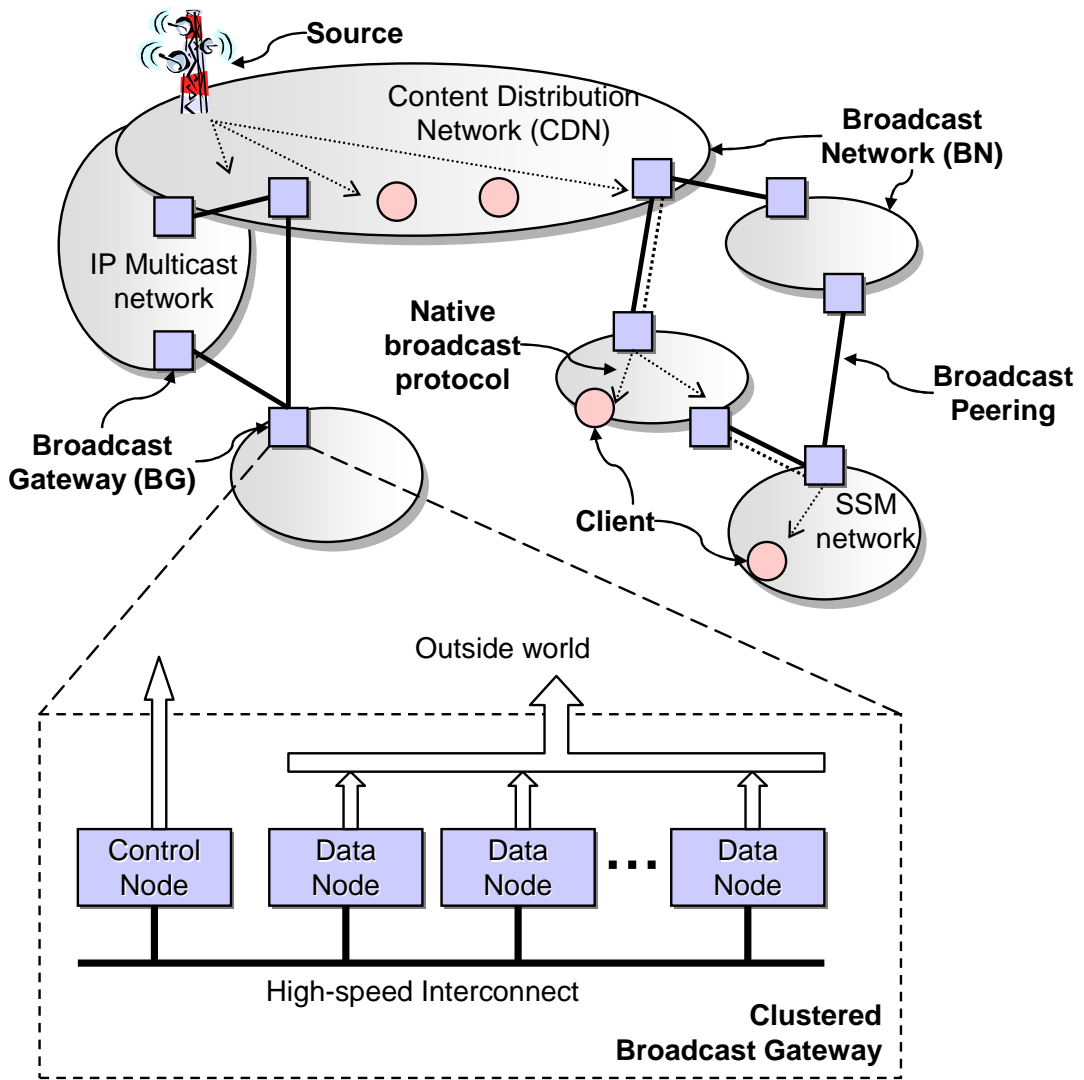


Figure 2: The Broadcast Gateway deconstructed

concerns that have confounded ISPs while deploying IP multicast technology. IP Multicast group addresses are a subset of the IPv4 address space and their scarcity is a well known problem. Distributed address allocation is likely to remain an issue even if IPv6 is widely deployed (which in itself is not a certainty). The open group semantic and lack of access control in IP Multicast, which allows any sender to send data to a group or a session is another area of concern. There is a range of business model concerns pertaining who to charge (among the receivers and the potentially many senders) and how to charge (considering factors such as total number of links used and the the rate of data being sent by each source). The state maintained at routers (per-source and per-group) is sometimes regarded as an obstacle to scalability. Furthermore, there is no well-accepted inter-domain solution. We will discuss this final point further in Section 2.1. Partly in response to these concerns, researchers and commercial developers have introduced new multi-point communication technologies that address some of the problems associated with IP multicast.

SSM (Source-specific Multicast) [4] limits the IP multicast service model from a many-to-many communication primitive to a one-to-many primitive. In doing so, it eliminates the problems of address allocation and access control that plague IP multicast. However, like IP multicast, SSM is network-layer technology, and although router support for SSM is available, few ISPs offer it as a widely-available service.

Recently, there have been proposals, from researchers and from content distribution companies, for moving the multi-point communication functionality away from network routers into application-layer overlays. The End-system Multicast project [17] builds multicast functionality entirely out of the end-hosts that participate in a multicast session. The Scattercast [7], Overcast [22], and Inktomi Broadcast Overlay [13] architectures build a broadcasting service via specialized broadcast servers that form overlay networks of unicast connections on top of the existing IP networks. By relying entirely on end-host deployment, commodity hardware and application-layer software, these systems make deployment substantially easier. However, there are limitations specific to these protocols. For example, the protocols used in End-system Multicast do not scale to groups larger than a few hundred receivers, and the Overcast architecture is designed specifically for single source bandwidth-intensive applica-

tions.

2.1 Related Work

To our knowledge, there has been no significant effort at bridging different multicast and broadcast networks into a single cohesive internetwork. Some content-distribution architectures such as Scattercast and the Inktomi Broadcast Overlay use a combination of unicast and IP multicast within their overlays, but they cannot interact with each other or with other broadcasting technologies, i.e., there is no mechanism for exchange of reachability information, tree-building information, or addressing information across networks implementing different broadcasting protocols. In the context of the web, we have seen some proposals [20, 5] for peering between independent Content Distribution Networks (CDNs). CDN peering provides CDNs a way to interoperate by allowing them to intelligently redirect clients to one another. Like CDN peering, the Broadcast Federation allows independent broadcast providers to interoperate. However, the distribution of live broadcast content across providers is quite different from the problem of redirecting web clients across CDN providers.

The Broadcast Federation problem is closely related to the inter-domain IP Multicast problem, which seeks to interconnect independent IP multicast-capable ISPs across the Internet. For this purpose, some multicast providers use Protocol Independent Multicast (PIM) [14] as the multicast protocol within their networks and the Multicast Source Discovery Protocol (MSDP) [24] to interconnect these individual networks. PIM contains a provision for sparse-mode operation which is based on receivers sending “join” messages toward a well-known Rendezvous Point (RP), which sets up a reverse-shortest-path tree rooted at the RP. In order to send data to a group, a source has to first send the data to the RP. This protocol is considered superior to protocols like DVMRP [34] because it avoids flooding the network for groups with sparse receiver sets, while the dense-mode behavior is similar for both protocols. MSDP interconnects different PIM networks with different RPs by flooding information about active sources to all RPs in the Internet. This limits the scalability of MSDP and its ability to be deployed globally.

The Border Gateway Multicast Protocol [23] improves upon MSDP by proposing a solu-

tion which is independent of the intra-domain multicast protocols. This solution is based on (a) defining the concept of a root domain to be associated with each multicast group, (b) using multi-protocol extensions to BGP [3] to distribute “group routes”, the shortest paths toward the root domain for each multicast group, and (c) building an inter-domain reverse-shortest-path tree based on explicit join messages toward the root domain for each multicast group. The Broadcast Federation architecture leverages these design ideas, but extends them beyond the IP multicast framework to apply to a heterogeneous collection of network- and application-layer broadcast technologies.

2.1.1 Server Architectures

Clustered server architectures have been extensively studied in the context of web servers [15, 26]. The benefits of using a cluster of commodity servers or workstations instead of customized hardware (typically used in routers and switches) include easier development and deployment of hardware and software, and easier masking of hardware or software failures (by replacing/repairing cluster workstations online while their load is being supported by the other workstations). On the other hand, the drawbacks of the clustered approach include the the large amount of space and power consumed when compared with customized hardware solutions.

The conventional cluster-server approach puts a front-end switching device between the Internet and a cluster of web servers, and attempts to balance the load among the web servers in the cluster, while trying to minimize request-response times and throughput. Every request (or packet) travels through the front end. However, in the Broadcast Gateway cluster, there is no directly analogous front-end - the data streams of each session need not flow through the control node. The control node only handles “control” traffic and no data traffic (this is explained in Section 4). Furthermore, much work on cluster-based services has focused on the maintenance of distributed state (with different levels of consistency). This is not an issue we address in our framework. The control node instantiates state for each session in only one data node at a time, and can always repeat the process for a different data node in case of data node failure. Therefore, consistency of state is not an issue in this context.

We note the possibility of failure of the control node. We do not address this issue in our work; we assume that traditional methods used in cluster-based services ([15]) can be used to mask such failures. Furthermore, while we have striven to develop as general a framework as possible, and while our work can be extended for use with short request-response applications (like typical WWW usage), our focus is biased toward applications which are (relatively) long-lived and high volume, like live video delivery. Finally, the placement of servers or server replicas has been studied in the context of the web [29]; we deem the analogous problem of placement of BGs in the Internet to lie outside the scope of this work.

3 The Federation Architecture

Figure 1 shows the essential components of the Broadcast Federation architecture. A federation is composed of a collection of *Broadcast Networks* or BNs that are interconnected via an overlay of *broadcast peering links* across application-layer *gateways*. We now define each of these elements.

3.1 Architecture Components

Broadcast Network (BN): A BN is a network which supports a single broadcast or multicast protocol natively and is bounded administratively and in scope. All nodes within a BN use the native broadcast medium to transmit or receive content local to their BN. The native broadcast medium may be based on either a network-layer protocol such as IP multicast [10] or source-specific multicast (SSM) [4], or an application-layer broadcast content-distribution network (CDN) [7, 13, 22]. A BN may be restricted to a single ISP's domain (as in the case of an AS), or may span multiple ISPs (as in the case of a CDN such as the Inktomi Broadcast Overlay [13]).

Each BN is identified by a globally unique 32-bit BN number (*BNNum*). This number needs to be assigned by some unique global entity like ICANN [18]. DNS can be used to convert descriptive BN names into BN numbers. For example, a well-known IP address within the BN can be used to represent its BN number, and an appropriate DNS name could be used

to map to the IP address. We note that the BN names and numbers are only for use with the Broadcast Federation protocols, and need not be recognized or used by each native broadcast protocol within the BN.

Broadcast Gateway (BG): The broadcast gateway is the point of peering between BNs, and implements the bulk of the Federation protocol suite. Each Broadcast Gateway has a module called *NativeNet* which implements the customization components that are required to allow the BG to communicate with its native broadcast medium and to translate incoming content and control messages into appropriate messages for the native medium. The BG appears as a single node to other elements in the architecture. It may be in practice be implemented as a single high end workstation, a specialized hardware solution based perhaps on a programmable router, or as a cluster of cooperating workstations [8, 15]. For reasons discussed in Sections 2.1.1 and 4 we have chosen to implement the BG as a cluster of workstations.

Broadcast Peering Link: A peering relationship between two BGs indicates a willingness to carry broadcast traffic to and from each other's networks. BGs across individual broadcast networks establish administratively configured peering links to each other (in contrast to self-organizing peers). This means that BN administrators establish explicit policies and commercial agreements for the use of those links, and configure their BGs accordingly. For example, BN administrators can configure peering links to allow or disallow the use of their network as a transit network for traffic to and from other reachable networks. Or, a peering link can be configured such that the rate at which traffic flows across it is bounded. While our implementation only supports simple policy filters based on incoming and outgoing peering links, nothing in our architecture prevents the usage of policy mechanisms similar to those found in BGP. Once the administrators have established their agreements and configured their BGs, the BGs communicate with each other over a reliable TCP connection and exchange reachability information. The nature of this reachability information is discussed further in Section 5.1. This TCP connection represents an "overlay link", i.e., it could traverse several physical links.

The peering connections that are set up across BNs are known as "external" peering. In

addition, all BGs within a single BN set up “internal” peering links with one another.

3.2 Service Model

The federation service model defines a broadcast *session* as a federation-wide group communication abstraction. Clients across the entire federation can subscribe to sessions published anywhere within the federation. Thus, a federation session extends an IP multicast network’s group abstraction or a single CDN’s stream beyond that network’s boundaries to reach clients in other BNs. Furthermore, a session may be composed of multiple *flows*. For example, a source can publish a federation-wide RTP session that is composed of a single RTP data flow and a separate RTCP control flow.

The network in which a broadcast session is originally published is designated the *owner BN* for that session. Typically, the owner BN is also the network in which the primary source of content for the session is located. For example, an online sports event that is broadcast via an ISP’s local IP multicast medium will choose that ISP’s network as the owner BN for the concert. Clients within the owner BN of a session can directly subscribe to the session without interacting with any of the federation components. This is to ensure that the native service model for a BN remains unaffected by the fact the the BN is part of a wider federation. Thus, in our online sports event example, clients within the owner ISP’s domain can tune into the event merely by subscribing to the IP multicast group for the concert. Beyond the owner BN, clients will rely on federation protocols to access the session via their broadcast gateways.

The owner BN and its broadcast gateways provide a convenient rendezvous point for the session. Other broadcast networks attach to the session via the owner BN’s broadcast gateways. Moreover, the concept of the owner BN lends itself to an intuitive naming scheme for broadcast sessions that avoids the address allocation problems associated with traditional multicast protocols. Across the federation, a session is identified by a combination of the owner BN’s BNum or its corresponding DNS name, and the native name of the session within the owner BN. We use a URL-like naming scheme for these federation-wide session names:

```
bfed://<owner-bn-name>/<native-name>/<flow-1-name>/<flow-  
2-name>/.../?<pmtr1>=<val1>&<pmtr2>=<val2>...
```

The `<native-name>` component of the session name represents the address or unique name associated with the session within the owner BN, and can be parsed only by gateways in the owner BN. Clients and gateways in other BNs simply regard it as an opaque string of characters. Furthermore, flows within a session are identified by components within the URL. Thus, the sports event originating within an ISP's IP multicast network will use the BNNum or DNS name associated with the ISP's network and the IP multicast group address (and port number) to form its federation-wide session name, for example:

```
bfed://multicast.isp.net/224.5.6.7/8888
```

In addition to the owner BNNum and the native name, a session URL may contain additional parameters. These parameters, which follow the “?” symbol in the URL, specify options for the session that allow the federation to support a wide range of session types and broadcast network capabilities without imposing a lowest-common-denominator service model across all networks. Thus, the federation can identify sessions as single-source or multi-source, real-time or bulk-data, etc. by specifying appropriate options within the session URL.

The entire session URL including the owner BNNum or name, the native session name, and the parameter list together form a unique session name across the federation. In our sports event example, assuming that content originates within an ISP's IP multicast network with RTP/RTCP data and uses ports 8888 and 8889 for the RTP and RTCP ports, the federation-wide session name is:

```
bfed://multicast.isp.net/224.5.6.7/8888/8889/?content=
RTP&single-source=true
```

We will now summarize the process of session establishment in the context of our sports event example. The source host performs whatever setup operations are required by its local multicast BN, for example, obtaining an IP multicast address for its use. Let us call this local BN ISP_1 and the IP Multicast address M . The source then publicizes the existence of the session, the nature of its content, and its federation-wide session name S , by listing it on its WWW page or a directory service (or is “found” by a search engine). It then transmits the

session data to the group M , within the network of ISP_1 . A client interested in this sports event “discovers” its existence and its federation-wide session name S by visiting the source’s WWW page or using a search engine or a directory service. If the client is not in the same BN as the source, it contacts a BG in its BN, and requests a stream for the the federation-wide session name S . This BG utilizes the federation protocols (discussed in detail in Sections 4 and 5) to obtain the stream from BGs in ISP_1 , which in turn obtain the stream by listening to the IP multicast group M . The details of the how a source advertises the existence of a session and its federation-wide session name, and of how a client discovers the same, are not within the scope of this document.

4 Design Overview

We now briefly outline the design of the Broadcast Gateways and the protocols that govern interaction between the gateways. The design of our protocols is guided by two key principles:

- Loosely-coupled interaction across BNs, rather than tight integration of each BN’s native broadcast protocols into the federation.
- The ability of each BN to publish sessions for its own clients without requiring any intervention from the federation. The federation protocols should be invoked only when clients need to access sessions across BNs.

If we consider each BG as a single atomic entity, the federation protocol suite is basically composed of three tiers: a routing layer that propagates reachability information across BNs, a tree-building layer that constructs distribution trees across the federation, and a data forwarding layer that handles the distribution of data packets across the dynamic trees constructed by the tree-building layer. These protocol layers allow a session to be disseminated via distribution trees that are constructed from the reverse shortest routing paths between each destination BN and the owner BN of the session, in a manner similar to that used by PIM [11] for transmitting IP multicast data.

The purpose of the routing layer is to enable each BG to find the next-hop BG along the shortest path toward all BNs. This ability is utilized by the tree-building component to build

reverse shortest-path distribution trees using a mechanism similar to that used in PIM-SM. Data is distributed along this distribution tree after the data forwarding component establishes data channels corresponding to each branch of the distribution tree. Each of the protocol layers has mechanisms that need to be customized for each specific BN and its local native broadcast medium. These customizations are encapsulated within a component called NativeNet (*Native Network* interface). This represents the interface between the federation protocols and the protocols of the native broadcast medium of each BN, and is discussed in detail in Section 5.4.

In such a system, there can be several limitations to scaling the number of sessions and bandwidth of the sessions: (i) the amount of control traffic, (ii) the routing convergence times, (iii) the bandwidths available between BGs and within BNs, and (iv) bottlenecks within the gateway itself. The first two limitations have been studied in the context of traditional routing protocols in the Internet. Limitation (iii) is addressed by our architecture to the extent that the best possible routes are selected. However, if no route is capable of carrying the offered load, the session cannot be transmitted at the desired bandwidth. It is possible to incorporate more advanced routing strategies based on multiple paths and similar techniques. However, that is out of the scope of this document.

Therefore, in this document, we focus on limitation (iv). The BG can become a bottleneck due to several reasons - (a) it cannot forward data packets at the desired rate due to I/O and CPU limitations, (b) it cannot handle the desired number of sessions due to resource constraints, or (c) it cannot handle the offered volume or rate of routing or tree-building traffic. While each of these problems can be addressed by specialized hardware design, or by buying better CPUs or more memory, we seek a general scalable solution. Problem (c) can be addressed by redundancy of the control node; as mentioned in Section 2, we do not address this problem here. We focus on problems (a) and (b).

We note that the processing of routing and tree-building messages (“control” traffic) is naturally decoupled from the forwarding of data traffic. The only interaction between the two functions is that the control traffic processing sets up per-session state in the BG which is used for data-forwarding. The state for each session is independent of the state for other sessions. Furthermore, it is quite likely that the volume and rate of data-traffic is much higher than that

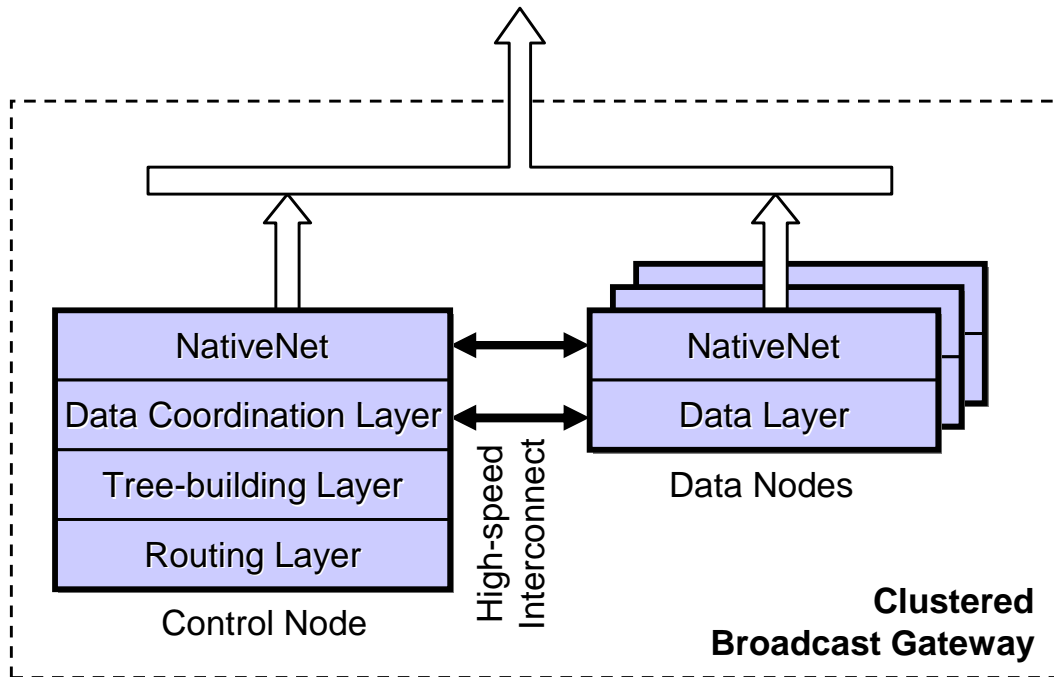


Figure 3: Components of a clustered broadcast gateway and the protocol layers within each component

of the control traffic. Hence we can distribute the processing load for different sessions across multiple nodes in a cluster. However the routing state at a BG cannot be easily divided on the basis of sessions³. This leads to the design of a BG as a cluster of workstations with a single control node, as shown in Figure 3.

A single BG cluster is composed of one control node and a number of data nodes. The control node implements the connectivity protocols *across* the federation, i.e., the aforementioned routing and tree-building layers. In addition the control node coordinates the distribution of data forwarding functionality across a number of independent data nodes. In other words, the data forwarding nodes handle the data forwarding functionality, and none of the routing or

³As mentioned in Section 2.1.1, we might actually want to divide or replicate the routing state across multiple control nodes in order to improve the availability or scalability of the control function. However, we do not consider this issue any further in this document, and assume that traditional methods in this realm ([15]) will suffice.

tree-building functionality. All nodes (control and data nodes) also implement the *NativeNet* interface, in order to adapt the policies and mechanisms in the routing, tree-building and data-forwarding layers to the protocols used in the specific BN and native broadcast medium.

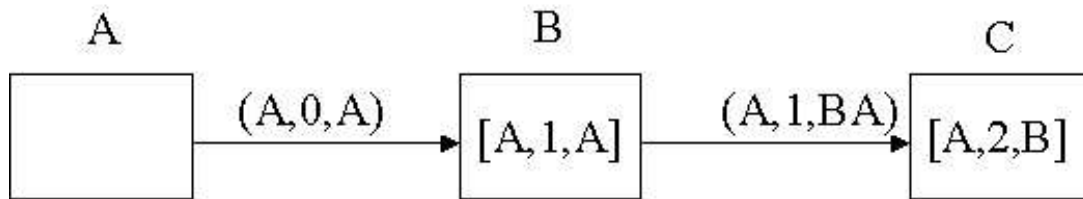
In summary, the control node is composed of four primary layers: a routing layer which propagates reachability information across BNs, a tree-building layer which constructs distribution trees across the federation, a data-coordination layer that manages the distribution of sessions across individual data nodes, and the *NativeNet* layer which is a customization layer for adapting the policies and mechanisms in the other three layers for the specific BN and native broadcast medium. Similarly, each data node is composed of two main layers: the data layer which handles the setting up of data forwarding paths and the distribution of data packets along these paths, and a customization *NativeNet* layer similar to that in the control node.

5 The Federation Protocols

We now describe in detail the different layers of functionality that reside in a BG: routing, tree-building, data-forwarding and *NativeNet*.


5.1 Routing Component

The routing component uses a path-vector protocol similar to BGP [30] to propagate information on how to reach other BNs to all the broadcast gateways in the system. Figure 4 shows the simplified operation of the path vector protocol for 3 gateways, using hop-count as the cost metric. Each broadcast gateway maintains a routing table that is used to compute the best route from a client's BN to any other BN in the federation. In order to ensure that per-session information is not sent to all BNs across the entire federation (many of which many not be interested in all of that information), we have chosen to make the routing component session-agnostic. That is, the routing layer should not distribute any information about individual sessions. This is key to ensuring that the routing infrastructure can scale independent of the total number of sessions.



(Destination, Cost, Path) – Route Advertisement

[Destination, Cost, Next-hop] – Routing Table

 – Broadcast Gateway

... The cost metric is hop-count...

Figure 4: Path Vector protocol for 3 gateways

The routing protocol actually advertises routes to individual BGs rather than to entire BNs, in order to obtain greater selectivity. This could be useful when a source- or owner-BN has more than one BG, and it might be desirable to choose the BG with a lower-cost path to the actual source host. We describe how this is achieved later, in Section 5.2.2. We note that the routing state maintained in each BG grows with the total number of BGs in the federation.

Traditional routing protocols use latency or hop count as the metric for evaluating routing costs. However, the routing metric that is most useful for a specific session depends on the characteristics of the session itself. For example, a delay-sensitive application like video conferencing cares most about the latency metric, while a bulk data transfer application needs routes that provide the best bandwidth performance. Hence, instead of fixing one routing metric for the entire federation, the routing layer presumes no single metric. Instead, it constructs a collection of routing tables, one for each metric of interest.

Different routing tables can also be used to encode the capabilities of the broadcast networks themselves. For example, BNs that are not capable of, or do not want to, support

multi-source broadcast distribution can opt out of acting as transit for such sessions by advertising independent routing tables for single-source and multi-source sessions. Thus a BN that uses a source-specific multicast (SSM) native medium can advertise infinite cost transit paths in its multi-source routing table, but advertise the actual routing costs in the single-source table. Similarly, we allow BNs to differentiate between their abilities to carry different kinds of traffic. For example, an HTTP-based CDN is not amenable to carrying real-time audio/video data over its TCP-based native broadcast medium, and hence can choose to advertise higher cost for such content.

Sessions indicate their requirements via parameters in their session URL. For example, `metric=latency,metric=bandwidth,multi-source=true,content=real-time,content=bulk-data`, etc. The tree-building layer described in the following section picks the appropriate routing table for its queries based upon these parameters that it extracts from the session URL.

The routing component provides hooks for implementing policies based upon peering agreements across BNs. For example, a BN may not wish to act as a transit network for other BNs. It can do this by applying appropriate filters to the routing information before forwarding it to its neighboring BNs. This is similar to the policy-based filtering that is used in the Internet today by BGP. The filtering hooks provide a basis for BN administrators to establish different kinds of commercial relationships with each other, like customer-provider relationships or peer-peer relationships.

In summary, the federation routing layer performs path-vector routing. It allows individual broadcast networks to export different routing costs depending upon individual session requirements and the capabilities of the BNs themselves. The routing information that is exported by a BN can be filtered based on administrative policies and peering agreements that the BN has set up.

The actual messaging protocol between two peer BNs relies on pairwise exchange of route-update messages over TCP connections. The first update conveys the entire routing table after being passed through policy-enforcing filters. Subsequent updates are incremental in nature. The BGs use periodic keep-alive messages to ensure that their peers are still available.

When a BG discovers the death of a peer due to loss of the keep-alive messages, it sends to all its other peers route withdrawal notifications for all the routes that went through the dead peer.

5.2 Tree-building Component

The tree-building component builds a shared inter-BN distribution tree for each session rooted at the owner BN of the session. All data traffic for the session flows along the shared tree. In our current framework, multi-source sessions do not construct per-source trees. The shared tree is constructed using the reverse shortest paths between the BGs of subscribing clients and the appropriate BGs within the owner network.

To join a broadcast distribution tree, clients must first communicate with a broadcast gateway within their BN. We introduce the concept of a *Mediator* which receives JOIN requests from clients and forwards them to an appropriate broadcast gateway. Once such an access gateway is located, the access gateway must initiate the tree-building process. In order to optimize distribution trees on a per-session basis, the tree-building process consists of two steps: a Session-specific Route (SROUTE) Discovery step, followed by a JOIN step. We look at each of these in detail now.

5.2.1 Mediator

When a client wishes to participate in a federation session, it needs to locate a broadcast gateway within its BN to forward its JOIN request to. However, the manner in which this is achieved in a scalable fashion depends upon the characteristics of the specific BN. Hence, we isolate this functionality into an abstraction that we call a *Mediator*. The mediator is responsible for intercepting JOIN requests from clients and forwarding them to one of the broadcast gateways within the client's BN. Each BN in the federation implements its own mediator. For example, in an IP multicast network, the mediator may be a well-known IP multicast group that clients and gateways subscribe to. Clients send their federation JOIN requests to this multicast group, the multicast routers forward the requests across the BN, and they get intercepted by the broadcast gateways. Alternatively, the BGs or the native broadcast

components such as a CDN's edge servers themselves can incorporate the functionality of the mediator. We describe more detailed examples of how mediators are specialized for specific broadcast networks in Section 6.

Clients in the owner BN of a session do not require any mediator. They simply use the owner BN's native subscription mechanisms to participate in the session. When clients need to access a session that does not originate in their own BN, they contact their local mediator and send it a JOIN request that includes the full session URL. The mediator in turn contacts an appropriate broadcast gateway to initiate the tree-building process. The broadcast gateway eventually responds with either a TRANSLATION message that contains a locally valid native broadcast address, or a NO_ROUTE message indicating that there is no path from the client's BN to the owner BN for the session. Clients can then directly subscribe to the native address specified in the *TRANSLATION* response.

In the above discussion, we have assumed that clients belong to or have access to a single BN, and are pre-configured with information about the local mediator for that BN. In such a scenario, clients may either be statically configured with this information or may be handed the information via special DHCP options. An alternative approach would be to establish a global mediator service that all clients contact. Such a mediator service would automatically determine the best access BN for each client out of potentially multiple choices, and then redirect the clients to the local mediator for that BN. However, the design of this global mediator service is orthogonal to the rest of the federation architecture, and is not addressed in this document.

5.2.2 Session-specific Route (*SROUTE*) Discovery

When a broadcast gateway receives a JOIN request from a client via the mediator, it needs to subscribe to the session via its best path to the owner BN. The JOIN request includes the session's URL name. Encoded within the URL are parameters that determine which routing metric to use for determining the best path toward the owner BN. The best path, however, may depend upon the location of the session source within the owner BN. Consider, as shown in Figure 5, an owner network (BN_1) with two broadcast gateways, *A* and *B*. Since the

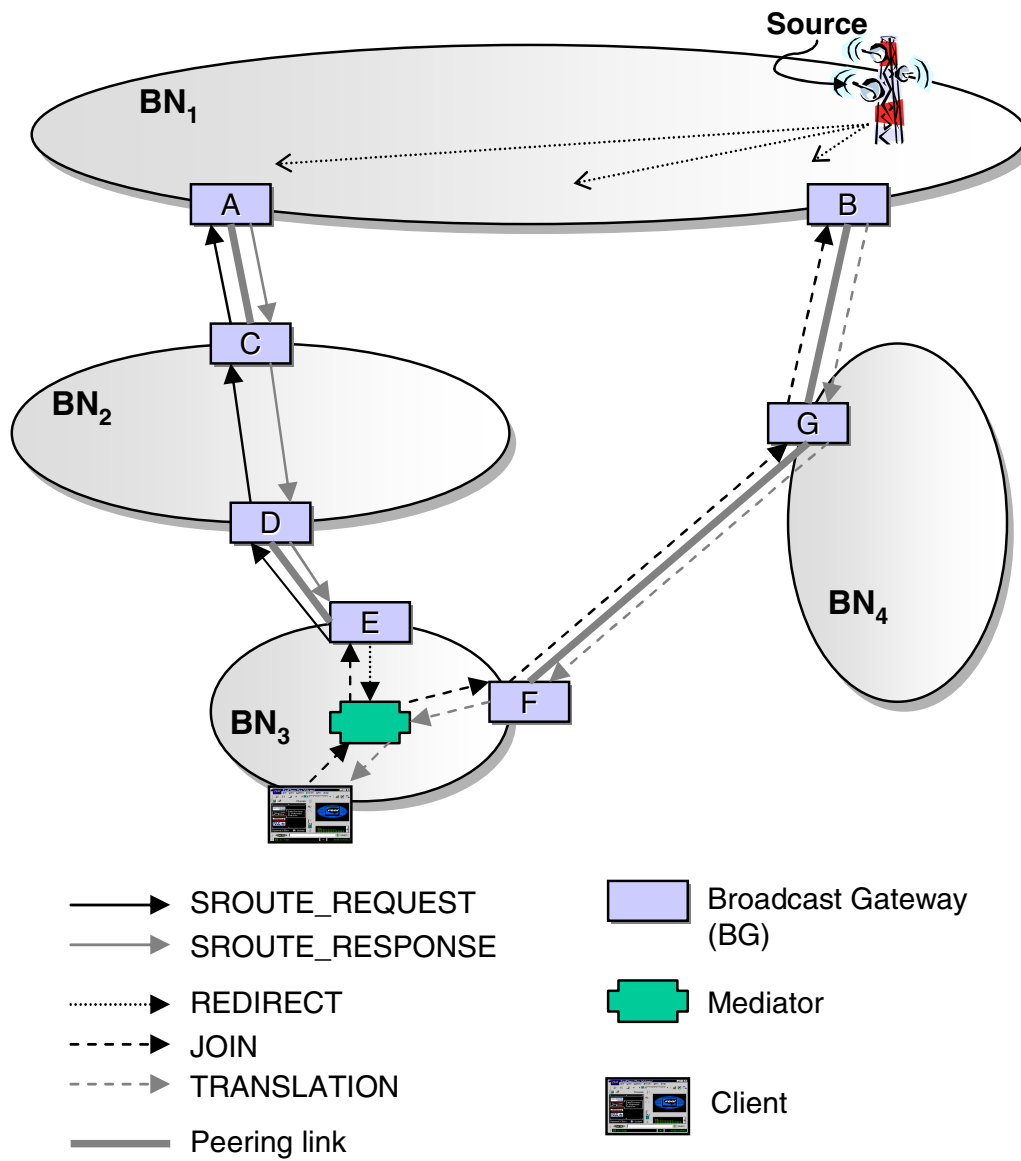


Figure 5: SROUTE messaging before JOINING

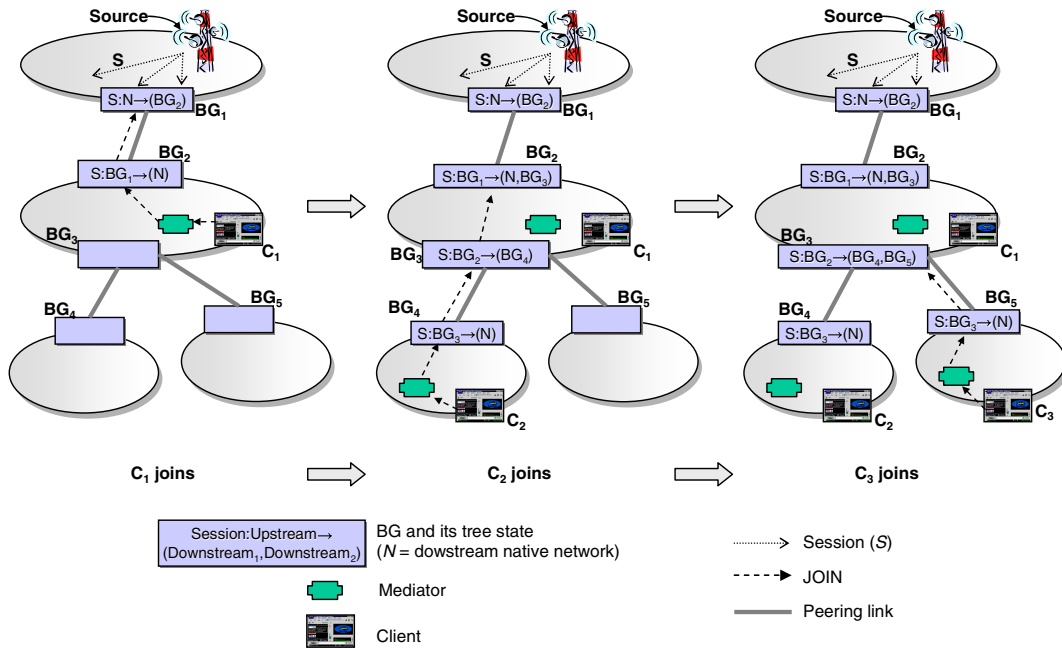


Figure 6: The tree-building protocol and the associated state maintained by BGs participating in the protocol.

routing layer keeps track of BG-BG routes rather than simply inter-BN routes, the broadcast gateway E in BN_3 has two routes $E - D - C - A$ and $E - F - G - B^4$ that lead to BN_1 . Similarly, BN_3 's other gateway, F , has two routes leading to BN_1 : $F - G - B$ and $F - E - D - C - A$. When a client in BN_3 joins a session via its local mediator, the mediator and the local gateways need to determine which of the above routes are best suited for accessing that session. This is achieved through the SROUTE Discovery protocol.

The discovery protocol works as follows. A mediator intercepts a JOIN request from a client and forwards it to any one of the broadcast gateways within its BN. In Figure 5, the mediator in BN_3 forwards a client's JOIN request for a session originating in BN_1 to its gateway E . E determines that there are multiple alternative paths that can be used to subscribe to this session. So it sends an SROUTE_REQUEST message along the better of its two paths leading to BN_1 . Included in this message is a list of broadcast gateways of BN_1

⁴This follows from the fact that BGs within a single network all set up internal peering connections to each other.

that E is aware of (i.e., A and B), that it discovered from the routing protocol. The message propagates along the path $E - D - C - A$. Once the **SROUTE_REQUEST** message reaches A , A needs to send back an **SROUTE_RESPONSE**. This response contains information about the proximity of the root of the native broadcast tree used within BN_1 to both the gateways A and B . For example, if BN_1 's native broadcast protocol constructs source-rooted trees, then the **SROUTE_RESPONSE** would contain the routing cost between the source and each of the gateways A and B . In order to determine these costs, A may have to internally communicate with the other gateway B to initiate cost measurement processes. How these costs are determined depends on the specifics of each individual BN, and is encapsulated within the NativeNet implementation for the BN. If a BN does not wish to expose any of its internals to the rest of the federation, its NativeNet implementation can simply return a null **SROUTE_RESPONSE**.

Once the **SROUTE_RESPONSE** returns to BN_3 , E can determine the best route for that specific session. It does so by appending the **SROUTE** costs to the BG-BG routing costs that it has stored in its routing tables. In our example, E determines that $F - G - B$ is the best session-specific route. So, it sends a **REDIRECT** response to the mediator indicating that it should subscribe to the session via F . At the same time, it forwards the **SROUTE** information to F . The mediator in turn sends a **JOIN** request to F which confirms that it is indeed the best route for this session by using the forwarded **SROUTE** information. Finally, F sends **JOIN** messages along $F - G - B$ to subscribe to the session. To avoid extra **SROUTE** discovery steps by BGs along the **JOIN** path, the **JOIN** message also forwards the **SROUTE** information that F already has.

All of this **SROUTE** information is soft state and needs to be periodically refreshed. Every time periodic **JOIN** messages reach the owner BN, it sends back **SROUTE** updates along the join path. Moreover, **SROUTE** state is maintained only along the path of broadcast gateways that care about it, that is, the set of gateways that are part of the distribution tree for the specific session. In our example, the **SROUTE** state is stored initially along the paths $A - C - D - E$ and $B - G - F$. However, C and D will eventually time out this state since the **JOIN** messages for this session (which cause the **SROUTE** state to get updated) flow along

the $F - G - B$ path. Other gateways in BNs that do not participate in this session will never receive or store the SROUTE information.

The SROUTE Discovery protocol is performed only when a downstream broadcast gateway has paths to at least two different BGs in the owner BN, and the next-hop peers for those paths are not the same. If there is a single unambiguous next hop for all paths leading from the downstream BG to the owner BN, it can directly send its JOIN message to this next hop peer without initiating any SROUTE discovery.

5.2.3 Tree Setup

Federation distribution trees are per-session shared trees rooted at the owner BN of the session. Downstream nodes send explicit JOIN messages along their best path toward the owner BN as computed by the SROUTE discovery process described above. Once a BG locates the appropriate route, it forwards the JOIN request to the next-hop BG along that route. The JOIN request is forwarded along next-hop BGs toward the owner BN until the request reaches a BG in the owner BN or a BG which is already part of the distribution tree for that session. If this final BG is in the owner BN, it then subscribes to the native session identified within the session URL.

Each BG along the distribution tree maintains tree state that includes the name of the session, the upstream peer in the tree leading toward the owner BN, and a list of downstream peers. Figure 6 shows an example of the tree-construction mechanism and the state that each BG maintains. This state is soft in nature and the BGs periodically update it by sending refresh JOIN messages upstream. When a BG stops receiving JOIN refreshes from any of its downstream peers, or if it receives an explicit LEAVE message, it removes the tree state associated with that peer. When it has no more downstream peers or clients for a session, it in turn sends a LEAVE message to its upstream peer. We note that the state maintained at each BG grows as the number of sessions being routed through that BG.

The routing and tree-building protocols discussed until now are handled solely by the control node within a BG. We now proceed to the protocols used in connection with the data nodes.

5.3 Data Path Setup

The control node's Data Coordination Layer in conjunction with the individual data nodes' Data Layers is responsible for setting up data communication *channels* for each of the separate flows within a session. A channel is an explicit communication link associated with a specific flow. All channels for a session are established along the edges of the distribution tree associated with that session. Across external peering links, channels use unicast communication, while within a BN or between internal peers, the forwarding channels use the native broadcast medium. Unicast channels can be either UDP for sessions with real-time or best-effort requirements, or TCP for bulk data transfer.

These data channels get set up via the exchange of TRANSLATION messages during the JOIN phase. The translation messages convert federation-wide session- and flow-names into locally valid names or addresses for each link along the distribution tree. As the JOIN request propagates upstream, each downstream BG assigns a data node and creates communication sockets within the data node for receiving data flows from its upstream peer. We now look at the protocol required for setting up the data channels along the tree.

5.3.1 Intra-BN Channel Setup

When the control node within a BG receives a JOIN request for a new federation session either from a local client (via the mediator) or from another internal BG, it notifies the data coordination layer of the JOIN request. The coordination layer must now assign this new session to one of the BG's data nodes to handle packet forwarding. It can do this in a simple round-robin fashion or use a more fine-grained load balancing technique that keeps track of the processing rates at each of the data nodes and assigns the new session to the least loaded node.

Once the control node picks a data node, the data node in conjunction with the data coordination layer on the control node sets up the data channels for all of the flows within the session. Figure 7 shows the steps needed to instantiate these channels. Upon receipt of a JOIN request, the control node C_A assigns a data node (say D_{A1}) for the new session and sends a `subscribe_down` message to it. This message contains the names of the session and

the flows within the session, along with a flag indicating that the subscription request came from within the BN. D_{A1} then allocates channels within its native broadcast medium for each of the flows in the requested session. For example, if BN_A is an IP multicast network, D_{A1} allocates a new IP multicast group address. For an SSM network, the data node would allocate a *(source, group)* address pair, while for a CDN network it would allocate an address (typically a CDN URL) that can be interpreted by the CDN's servers. The mechanism for allocating a new broadcast channel is encapsulated within the *NativeNet* layer, which we will describe in Section 5.4. D_{A1} returns this allocated address (multiple addresses if there is more than one flow) back to the control node which in turn sends a TRANSLATION response to the requesting client (via the mediator). The client can then subscribe directly to the address included in the TRANSLATION message.

5.3.2 Inter-BN Channel Setup

Across BGs in different broadcast networks, the data channel setup works as follows. Before forwarding the client's JOIN request to the upstream BG, C_A sends **allocate** requests to D_{A1} for each of the flows within the session. The request contains the session- and flow-names. Based on the parameters associated with the session (for example, real-time vs bulk-data), D_{A1} determines the kind of transport protocol required for receiving the flow from the upstream BG and accordingly allocates a data channel for it. For example, to receive a real-time flow from an external peer via unicast, the data node will allocate a unicast UDP socket. Depending upon the implementation of the data node, it may either allocate a new socket for each flow that it handles, or reuse the same one for all UDP-based flows. D_{A1} returns the address (N_1) of the socket back to the control node as part of the **allocate_response** message. This address is called the *translation* since it is essentially a mapping between a federation session and flow name to a native IP communication address. The control node C_A forwards N_1 to the upstream BG by piggybacking it onto the JOIN message.

When the control node C_B in the upstream BG receives the JOIN message from its downstream peer, it in turn allocates a data node within its cluster (say D_{B1}) and sends it a **subscribe_down** request. Like D_{A1} , D_{B1} allocates a data channel for the new flow. In addition,

D_{B1} assigns a locally unique 32-bit flow label to the flow. Later on, when it forwards packets from this flow to the downstream nodes, it includes this flow label in a packet header. The downstream nodes can use the flow label for fast lookup within their forwarding tables (rather than indexing the tables on the variable length session URLs). D_{B1} returns the translation N_1 associated with the newly allocated data channel and the flow label L back to the control node, which in turn sends them on to the downstream BG. Then, the downstream control node C_A sends a `subscribe_up` message to the data node D_{A1} associated with the session. Finally, D_{A1} completes the channel setup for communication between N_1 and N_2 . For example, if the channel is a TCP connection, it connects to the TCP address/port mentioned in the translation N_2 returned by the upstream data node.

5.3.3 Data Forwarding

Forwarding of data packets takes place along the channels as they get set up along with the `JOIN` messages that propagate up to the owner BN. Distribution trees are assumed to be uni-directional unless a session is specifically flagged as multi-source (via a parameter in the session URL). For uni-directional trees, a BG forwards a packet to all of its downstream tree neighbors only if it was received from the BG's upstream neighbor in the tree. For multi-source sessions, the trees are bi-directional and a BG forwards packets received from any of its tree neighbors to all of its other tree neighbors. Across external links, flow labels allow for fast lookup of entries within the forwarding table.

Given our description of the data path setup, there can be several possible bottlenecks or limitations in the forwarding paths of each data node: the network interface, the CPU processing power, the I/O processing libraries and interface provided by the OS, user-level implementation (as opposed to a kernel-level implementation), etc. Different performance results will be obtained for different combinations of the above factors; this can only be determined experimentally. There can also be a limitation on the number of sessions supported by each data node if the OS imposes a limit on the number of file descriptors or the memory used (mostly for forwarding path information and data buffers).

5.3.4 Soft State

Like the tree-building state, all of the data channels rely on soft state. The periodic JOIN messages generated by the tree-building layer refresh the data channels. If the JOIN messages stop arriving, or if an explicit LEAVE message is received, the channel state is torn down. Moreover, if the routing state changes, resulting in a corresponding change in the distribution tree, new JOIN messages received along the new distribution paths will automatically trigger the set up of new data channels for the updated tree. Similarly, if a data node crashes, the data-coordination layer notices this and automatically allocates a new data node for the session thereby masking any partial failures within the BG cluster. We note that the state maintained at a data node grows as the number of sessions handled by that data node.

5.4 NativeNet

In our prior discussion, we have assumed that the various components of the broadcast gateway know how to communicate with the gateway's native network, for example, allocating addresses in the native broadcast medium, sending and receiving packets on the native medium, etc. Rather than expose each broadcast network's functionality to all of the other components of the BG, we encapsulate this functionality in a simple and explicit customization API that we call the Native Network Interface or *NativeNet*. The NativeNet layer can be implemented as a dynamically loadable library that is loaded into the control node as well as all of the data nodes at run-time. In the control node, the NativeNet provides the interface to the native mediator for receiving or recognizing federation JOINS and LEAVES. In the data nodes, the NativeNet allocates native broadcast addresses, subscribes native broadcast channels, sends and receives broadcast data, etc.

Figure 8 shows the API for the NativeNet layer. Each broadcast network must provide its own implementation of this API. The NativeNet API consists of seven major downcalls that various components of the BG make to the NativeNet implementation, and three main upcalls (or notifications) from the NativeNet to the other BG components. We describe these briefly below:

```

class NativeNet {
public:
    /* Data Node downcalls */
    void *allocate_channel(SessionDescription *session,
        char *flow_name, Boolean up_or_down,
        char *&ret_addr, int &ret_len);
    void subscribe(SessionDescription *session,
        char *flow_name, Boolean up_or_down,
        void *channel, char *addr, int len);
    void refresh(SessionDescription *session,
        char *flow_name, Boolean up_or_down,
        void *channel);
    void unsubscribe(SessionDescription *session,
        char *flow_name, Boolean up_or_down,
        void *channel);
    void reclaim_channel(SessionDescription *session,
        char *flow_name, Boolean up_or_down,
        void *channel, char *addr, int len);
    int send_data(void *channel, char *data, int length);
    List *get_sroutes(SessionDescription *session,
        List *route_requests, Boolean *got_all_ptr);

    /* Data Node upcall */
    void recvd_data(void *channel, char *data, int length);

    /* Control Node upcalls */
    void recvd_join_request(SessionDescription *session);
    void recvd_leave_request(SessionDescription *session);
}

```

Figure 8: NativeNet API

- **allocate_channel:** Invoked when the data node needs to allocate a native (local) broadcast address for a new federation session.
- **subscribe:** Subscribe to the native broadcast channel that was allocated for a specific federation session, i.e., listen for arriving data packets on that channel and hand those data packets back to the data forwarding component.
- **refresh:** Invoked periodically to allow the NativeNet to refresh its subscription to the native broadcast channel if necessary.
- **unsubscribe:** Invoked when the channel is no longer needed.
- **reclaim_channel:** Invoked to allow the NativeNet implementation to reclaim previously allocated channel addresses that are no longer needed.
- **send_data:** Used to send data packets to the native broadcast medium.
- **get_sroutes:** Invoked to retrieve SROUTE information from the owner BN.
- **recvd_data:** Invoked by the NativeNet in the data node whenever it receives data from a native channel that it had previously allocated.
- **recvd_join:** Invoked by the NativeNet in the control node whenever it receives a JOIN request from a client via the local mediator. Depending upon the NativeNet implementation, the mediator functionality may be part of the NativeNet itself.
- **recvd_leave:** Invoked by the NativeNet in the BG's control node when it no longer has clients within the broadcast network that are interested in a previously subscribed session.

Here is an example of how this NativeNet API is customized for specific BNs: in this case an IP multicast BN. The IP multicast NativeNet uses a pre-assigned pool of IP multicast group addresses to allocate to data channels for requested federation sessions ⁵. The `allocate_channel()` call assigns an unused address from this pool. The `subscribe()` call opens a

⁵This is a rather simplistic method of address allocation. More advanced methods of address allocation similar the method in [23] can be used here.

datagram socket and makes that socket subscribe to the allocated IP multicast group. Since the IP multicast subscription API automatically deals with refreshing the multicast join state, the `refresh()` call has an empty implementation. `unsubscribe()` closes the multicast socket, and `reclaim_channel()` returns the assigned multicast group address back to the unused pool of addresses. The `send_data()` call is used to send data to the native IP multicast group. If a data packet is received from an IP multicast group (that the NativeNet component had earlier subscribed to), the packet is handed to the data-forwarding component by the `recvd_data()` upcall.

The multicast NativeNet implementation uses a well-known IP multicast group to support the mediator functionality. This mediator multicast group for this BN needs to be “well-known” only within the BN. Clients send damped JOIN requests for federation sessions to the mediator group, and BGs’ control nodes respond to those requests with TRANSLATION messages containing the IP multicast group address that was allocated for the session in the JOIN request. If multiple BGs within the BN can respond to a JOIN request, the BG with the shortest (SROUTE-augmented) path to the owner network wins. This mediator functionality is used to indicate to the NativeNet component that a JOIN or LEAVE is requested, which triggers a `recvd_join()` or `recvd_leave()` downcall to the BG’s tree-building component.

6 Experiments and Results

We have built a prototype implementation of the clustered broadcast gateway in the form of event-driven single-threaded application-level programs written in C++. We have implemented custom NativeNets for IP multicast, Source-specific Multicast (SSM), and a simple HTTP-based content-distribution network (CDN). Each of these NativeNet implementations has a code complexity of around 500-700 lines of C++ source code. Even though these NativeNet implementations are bare-bone, their code sizes still indicate that it is easy to customize our system for deployment in a BN with a particular broadcast protocol.

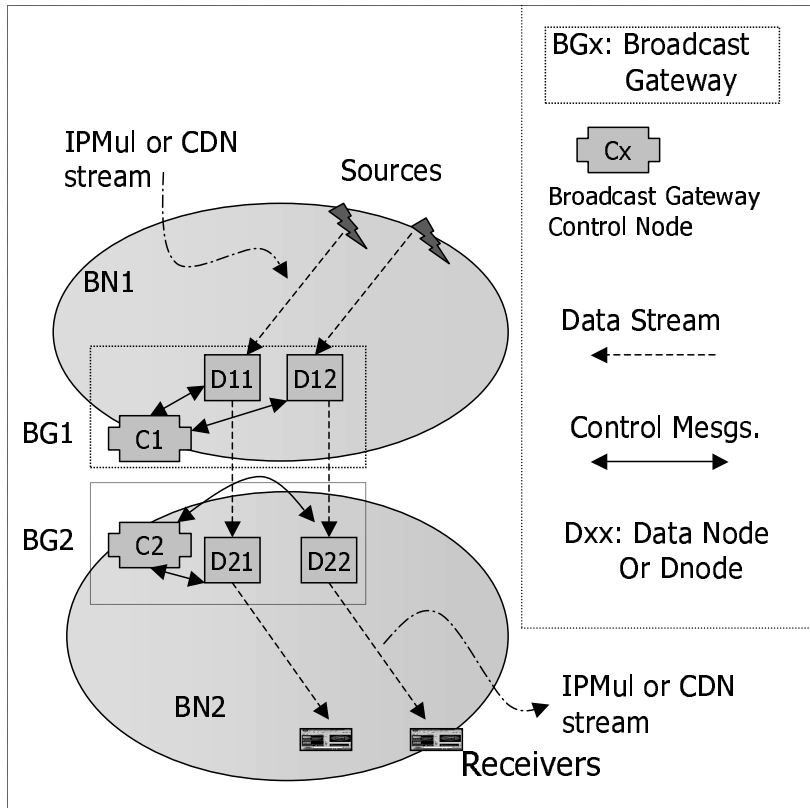


Figure 9: Experimental setup of clustered gateways

6.1 Experimental Setup

We performed several experiments with different BN topologies to test the correctness of our protocol implementations. However, we do not quantitatively investigate properties of the routing and tree-building protocols such as routing stability, convergence times, and impact of control traffic volume, since these properties have been studied in the context of similar protocols (e.g., BGP) in the Internet ([16],[31],[6]). Therefore we focus on the throughput- and session- scalability of the clustered BG implementation.

We performed our experiments on the Millennium [32] test-bed. The machines in the test-bed were (unless otherwise specified) 496MHz Pentium III PCs with 1Gbps network interface cards, running Linux (kernel version 2.4.18). The machines were connected to two switches capable of forwarding packets at a rate greater than 50Gbps. The two switches were interconnected via an 8Gbps link. On top of this test-bed, we tested the ability of the broadcast gateways to sustain high throughput and to serve a large number of simultaneous sessions.

For all of our experiments, we used the setup shown in Figure 9. The setup consists of two BNs each with one broadcast gateway. The two BGs are connected via a peering link. Data sources are located in BN_1 . Clients in BN_2 attempt to access the data generated by these sources. Mediators within BN_2 send the clients' JOIN requests to BG_2 . Upon completion of the data channel setup, the packets generated by sources in BN_1 are received by a data node in BG_1 , forwarded to a data node in BG_2 , and finally forwarded to the clients. The clients receive this data and report the observed throughput. The reason for using this setup and this testbed was to ensure that the data nodes in the BGs were the bottlenecks in the data stream, and further, that the bottleneck was not due to a low-bandwidth I/O interface or switch. Note that this testbed is not appropriate for latency-sensitive testing since it does not accurately reflect wide-area latencies.

Our experiments assume that each of the BNs is a CDN which distributes high volume content for long-lived sessions using the HTTP protocol. Similar experiments can be run for IP multicast-based networks. However, at high data rates, in the absence of adequate flow- and congestion-control, the UDP packet handling breaks down and results in unacceptable levels of packet loss. Hence, to perform realistic scalability experiments for multicast, we

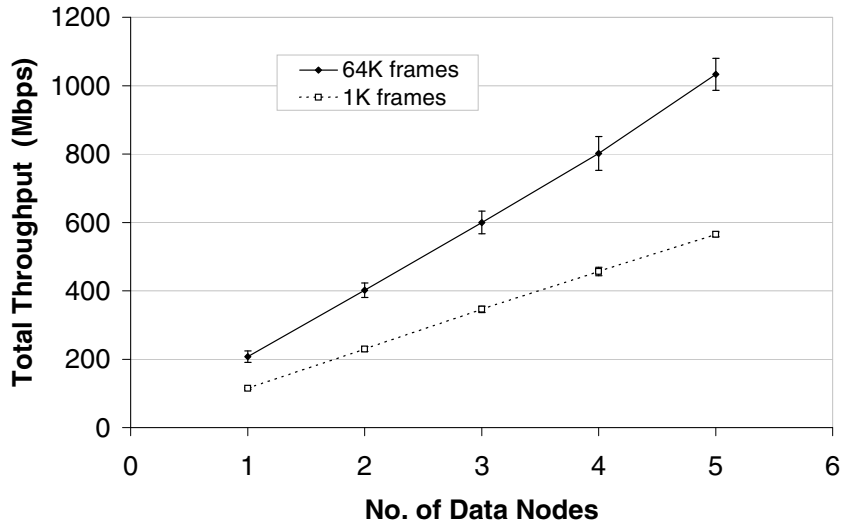


Figure 10: Single session throughput for different numbers of data nodes.

would have to implement real-time flow- and congestion-control for UDP. This is an area for possible future work.

The first quantity we observe is the maximum throughput that a BG can sustain while forwarding data between two BNs. This is affected primarily by the processing capability within the BG and its I/O interfaces. We note that the primary bottleneck for throughput is the data traffic. The overhead associated with control traffic is minimal. At refresh time periods of 30 seconds, with 1 KB (maximum) packets used to refresh state at the data nodes, the control traffic would form less than 0.3% by volume of a 100Kbps data stream. At these ratios, the control traffic does not significantly affect the forwarding throughput. Another aspect to observe is the behavior of the data forwarding as we scale the number of sessions in the system. All of our results are averages of five runs of three minutes each. Where depicted, confidence intervals are for 95%, obtained across these runs.

6.2 Results

We measured the maximum throughput that a BG could sustain while serving a very small number of sessions, with different numbers of data nodes available to each BG. The sources transmitted data at as high a rate as can be sustained by their TCP connections. There was

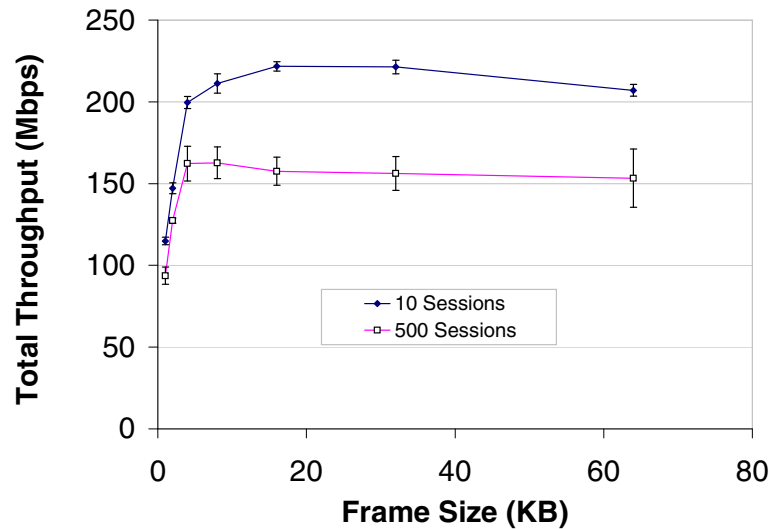


Figure 11: Single-session throughput through a single data node, for different data frame sizes.

one client and one source machine used per data node available to each BG, in order to ensure that any bottlenecks observed would be caused by the BGs. The number of sessions used was one per source (and client). Figure 10 shows the results of this experiment. The two lines correspond to two different data frame (or application data unit) sizes used. First, we note the almost linear increase in throughput as data nodes are added. This demonstrates the ability to scale throughput incrementally by adding data nodes, for this range of cluster size. Second, we note that the BG can forward around a gigabit per second with 5 data nodes. In fact, with a more powerful set of machines—700 MHz Pentium III-based CPUs—we only required 4 data nodes to achieve such speeds.⁶ Third, we note that the smaller frame size leads to a lower forwarding speed. The reason for this is that the processing at the data node, involving context switches (note that this is a user-level implementation), system calls, polling, and table lookups, has to occur more often for smaller packets. This is explored in more detail in Figure 11, which shows the variation of throughput with frame size. This particular experiment was performed with one data node per BG, and one session.

Next, we measured the maximum throughput that a BG can sustain while varying the

⁶These faster machines were not available for any of the other experiments.

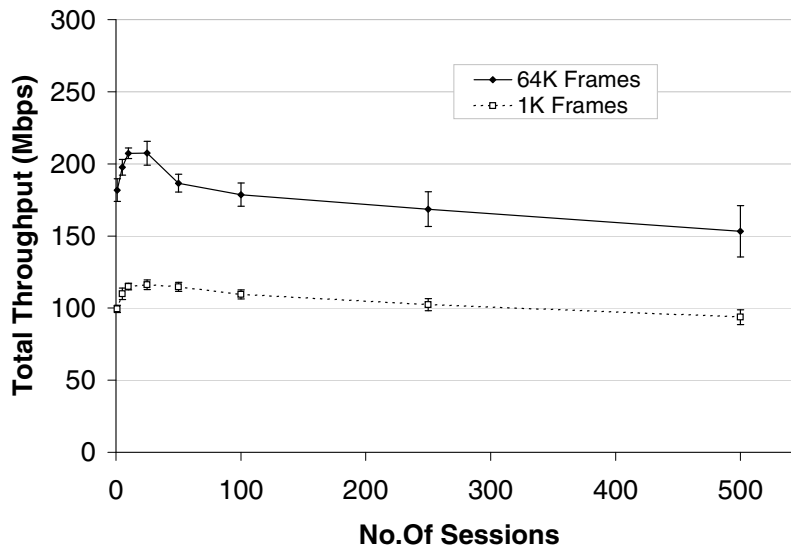


Figure 12: Multiple session throughput with 1 data node.

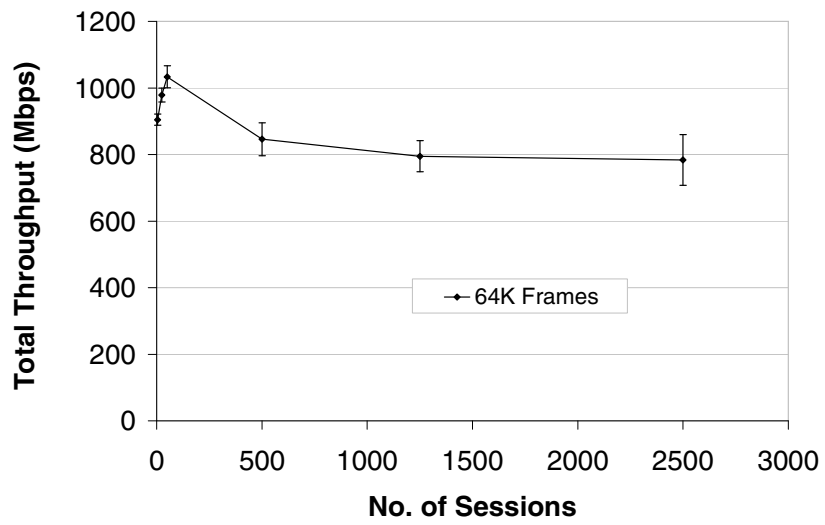


Figure 13: Multiple session throughput with 5 data nodes.

number of sessions. This was done in two scenarios: with one data node per BG and with 5 data nodes per BG. Figures 12 and 13 show the results of these two experiments respectively. We notice that the aggregate throughput across all sessions remains high but exhibits a slow decrease. This means that, subject to a loss factor, each session is capable of harnessing as much of the available bandwidth as is possible given the limitations of the network and the processing power of the BGs. The main reason for the loss factor is scaling behavior of the GLib event-loop library used by our implementation which in turn utilized the `poll()` interface. Profiling of the program behavior revealed that the set of functions involved in marshaling a linear array of arguments for the `poll()` call, and subsequent checking for return value flags consumed significantly more time when the number of sessions (and therefore, file descriptors) was increased from 1 to 500. These problems can be alleviated by using faster implementations for network I/O, such as the FreeBSD *kqueue* system call or the IO-Lite kernel [27]. Each data node could handle upto 509 sessions. This was the limit on the number of sessions, because each session involved the use of 2 sockets, in addition to a few common sockets, and the total number of open file descriptors on our test-bed systems was limited to 1024.

Finally, in order to demonstrate a situation with low offered load, we used rate-limited sources whose rates were approximately 100Kbps in order to examine the behavior of the BGs as the number of sessions increased but the traffic volume remained low. We noticed that the throughput remained essentially constant at 100Kbps as the number of sessions used was increased. This implied that the per-packet forwarding functionality within the BG was independent of the number of simultaneous sessions at low volumes.

To summarize the important results, we observed that the single session throughput scaled linearly (a little less than 200Mbps per data node) with the number of data nodes when the number of data nodes was between one and five. The throughput (single-and multiple-session) increased with increasing frame sizes, and saturated at around a frame size of 16KB). The throughput observed decreased when the number of sessions increased beyond a few 10s, primarily due to the scaling behavior of the I/O interface used. The primary bottleneck was the CPU processing power (given that we used 1Gbps NICs). The limitations which caused the

observed graphs were basically due to memory copies, context switches, and the I/O interface used. In addition there was a limitation on number of file descriptors, which limited the maximum number of sessions per data node to around 500.

7 Conclusion and Future Work

In this report, we have proposed an architecture that constructs a federation of broadcast networks via an overlay that is composed of application layer broadcast gateways. Our architecture can be easily customized to a range of network-layer and application-layer broadcast networks. We have described our design of a high performance clustered implementation of a broadcast gateway, and presented performance results for this clustered gateway. These results are promising, for example, the ability to scale to gigabit speeds and the ability to support 2500 sessions with only 5 data nodes.

While our work serves to demonstrate the feasibility of inter-domain protocol-independent broadcast, improvements are possible and several impediments remain. These are possible areas for future work.

Our current architecture presumes that clients know which access BN they belong to and how to contact the mediators within that BN. It would be useful to design a framework for automatically choosing an appropriate access BN (from potentially multiple overlapping choices), thereby providing a seamless mechanism for clients to access broadcast sessions anywhere on the Internet. One simple method would be to assume that each client is configured with the address of at least one mediator or BG. This mediator or BG can then give the client a list of every other mediator in each BG that the client specifies (that it has direct access to). The client can then query each mediator in the list for the cost of its path to the session's owner BN, and choose the one with the lowest total cost. For popular sessions, this would introduce a large amount of query traffic. Therefore, it appears that it would be more promising to set up 'probe' machines which would obtain the required performance information as outlined above, and then cache it for use by clients situated nearby (topologically). It would also be possible to logically co-locate the probe nodes and the mediator functionality. The probe nodes would have to be chosen such that they reflect the same performance to the

Internet as the set of clients they serve, using methods similar to those used in [21].

Our architecture's routing protocol is similar to BGP, and therefore shares with it problems related to stability and configuration-intensiveness, which are under study in the context of BGP. The protocol would benefit from inclusion of mechanisms that improve its stability in the face of perceived peering topology changes, such as those discussed in [33] (Route Flap Dampening) and [1].

The current implementation is a user-level implementation. It would be possible to demonstrate significantly better performance using a kernel level implementation. One way to achieve this is to leverage the Click modular router [25], which is capable of kernel mode operation. In the absence of a kernel level implementation, performance benefits can still be achieved by using a different Linux kernel version or a kernel modification to support a more efficient event-delivery interface, like the mechanism used in [2], or the “%dev/poll” interface ([28]).

Finally, deployment in a realistic Internet setting would yield greater insight into the performance of our architecture. There are two main impediments to widespread deployment (in a non-commercial fashion): (i) it is hard to obtain machines (and clusters of machines) located in different parts of the country or world (ii) even protocols that we envision will work well within BNs have not been significantly deployed, or are proprietary in nature. It appears that at least the first problem can be alleviated by a distributed testbed like PlanetLab [19].

8 Acknowledgments

This work was done under the guidance of Prof. Randy Katz (UC Berkeley, CA), and in conjunction with Yatin Chawathe (ATT Research, Menlo Park, CA). I thank Eric Fraser (UC Berkeley, CA) for his help with the Millennium cluster of machines. I also thank Prof. Katz and Prof. Stoica for reading this report and for their suggestions regarding the same.

References

- [1] BALLARDIE, T., FRANCIS, P., AND CROWCROFT, J. . In *Proceedings of ACM SIG-*

COMM '93 (San Francisco, CA, Aug. 1993).

- [2] BANGA, G., MOGUL, J. C., AND DRUSCHEL, P. A scalable and explicit event delivery mechanism for UNIX. In *USENIX Annual Technical Conference* (June 1999), pp. 253–265.
- [3] BATES, T., CHANDRA, R., KATZ, D., AND REKHTER, Y. *Multiprotocol Extensions for BGP-4*, Feb. 1998. RFC-2283.
- [4] BHATTACHARYYA, S., DIOT, C., GIULIANO, L., ROCKELL, R., MEYLOR, J., MEYER, D., SHEPHERD, G., AND HABERMAN, B. *An Overview of Source-Specific Multicast (SSM) Deployment*. Internet Engineering Task Force, Mar. 1999. Internet Draft, draft-ietf-ssm-overview-02.txt.
- [5] BILIRIS, A., CRANOR, C., DOUGLIS, F., RABINOVICH, M., SIBAL, S., SPATSCHECK, O., AND STURM, W. CDN Brokering. In *Proceedings of the Sixth International Workshop on Web Caching and Content Distribution* (Boston University, Boston, MA, June 2001).
- [6] BILLHARTZ, T., CAIN, J. B., FARREY-GOUDREAU, E., FIEG, D., AND BATSELL, S. G. Performance and resource cost comparisons for the CBT and PIM multicast routing protocols. *IEEE Journal on Selected Areas in Communications* 15, 3 (1997), 304–315.
- [7] CHAWATHE, Y. *Scattercast: An Architecture for Internet Broadcast Distribution as an Infrastructure Service*. PhD thesis, University of California, Berkeley, Dec. 2000.
- [8] CHAWATHE, Y., AND BREWER, E. System Support for Scalable and Fault Tolerant Internet Services. In *Proceedings of Middleware '98* (Lake District, U.K., Sept. 1998).
- [9] CNN NEWS. Millions watch Madonna online, Nov. 2000. <http://www.cnn.com/2000/SHOWBIZ/Music/11/28/britain.madonna/>.
- [10] DEERING, S., AND CHERITON, D. Multicast Routing in Datagram Internetworks and Extended LANs. *ACM Transactions on Computer Systems* 8, 2 (May 1990), 85–110.

- [11] DEERING, S., ESTRIN, D., FARINACCI, D., JACOBSON, V., LIU, C.-G., AND WEI, L. An Architecture for Wide-area Multicast Routing. *IEEE/ACM Transactions on Networking* 4, 2 (Apr. 1996).
- [12] DIOT, C., LEVINE, B. N., LYLES, B., KASSAN, H., AND BALENSIEFEN, D. Deployment Issues for the IP Multicast Service and Architecture. *IEEE Networks special issue on Multicasting* (2000).
- [13] FASTFORWARD NETWORKS/INKTOMI MEDIA DIVISION. Broadcast Overlay Architecture, 2000. <http://www.ffnet.com/>.
- [14] FENNER, B., HANDLEY, M., HOLBROOK, H., AND KOUVELAS, I. *Protocol Independent Multicast–Sparse Mode (PIM-SM): Protocol Specification (Revised)*. Internet Engineering Task Force, draft-ietf-pim-sm-v2-new-05.txt, Mar. 2002. Internet Draft.
- [15] FOX, A., GRIBBLE, S., CHAWATHE, Y., BREWER, E., AND GAUTHIER, P. Cluster-based Scalable Network Services. In *Proceedings of SOSIP '97* (St. Malo, France, Oct. 1997), pp. 78–91.
- [16] GRIFFIN, T. G., AND PREMORE, B. J. An Experimental Analysis of BGP Convergence Time. In *Proceedings of the 9th International Conference on Network Protocols* (Lancaster, U.K., Nov. 2001).
- [17] HUA CHU, Y., RAO, S., AND ZHANG, H. A Case For End System Multicast. In *Proceedings of ACM Sigmetrics '00* (Santa Clara, CA, June 2000).
- [18] The Internet Corporation for Assigned Names and Numbers. <http://www.icann.org>.
- [19] INTEL RESEARCH. The PlanetLab Testbed. <http://www.planet-lab.org>.
- [20] INTERNET ENGINEERING TASK FORCE. IETF Content Distribution Internetworking Group. <http://www.content-peering.org/ietf-cdi.html>.
- [21] JAMIN, S., JIN, C., JIN, Y., RAZ, D., SHAVITT, Y., AND ZHANG, L. On the placement of internet instrumentation. In *INFOCOM (1)* (2000), pp. 295–304.

- [22] JANNOTTI, J., GIFFORD, D. K., AND JOHNSON, K. L. Overcast: Reliable Multicasting with an Overlay Network. In *Proceedings of the 4th Symposium on Operating System Design and Implementation (OSDI)* (San Diego, CA, Oct. 2000), USENIX.
- [23] KUMAR, S., RADOSLAVOV, P., THALER, D., ALAETTINOGLU, C., ESTRIN, D., AND HANDLEY, M. The MASC/BGMP Architecture for Inter-domain Multicast Routing. In *Proceedings of SIGCOMM '98* (Vancouver, British Columbia, Canada, Sept. 1998).
- [24] MEYER, D., AND FENNER, B. *Multicast Source Discovery Protocol (MSDP)*. Internet Engineering Task Force, `draft-ietf-msdp-spec-13.txt`, Nov. 2001. Internet Draft.
- [25] MORRIS, R., KOHLER, E., JANNOTTI, J., AND KAASHOEK, M. F. The click modular router. In *Symposium on Operating Systems Principles* (1999), pp. 217–231.
- [26] PAI, V., ARON, M., BANGA, G., SVENDSEN, M., DRUSCHEL, P., ZWAENEPOEL, W., AND NAHUM, E. Locality-Aware Request Distribution in Cluster-based Network Servers. In *Proceedings of the Eighth International Conference on Architectural Support for Programming Languages and Operating Systems* (San Jose, CA, Oct. 1998), ACM.
- [27] PAI, V., DRUSCHEL, P., AND ZWAENEPOEL, W. IO-Lite: A Unified I/O Buffering and Caching System. In *Proceedings of the 3rd Symposium on Operating System Design and Implementation (OSDI)* (New Orleans, LA, Feb. 1999), USENIX.
- [28] PROVOS, N., AND LEVER, C. Scalable Network I/O in Linux. In *In FREENIX track USENIX Annual Technical Conference* (June 2000).
- [29] QIU, L., PADMANABHAN, V. N., AND VOELKER, G. M. On the Placement of Web Server Replicas. In *Proceedings of INFOCOM 2001* (Anchorage, Alaska, Apr. 2001).
- [30] REKHTER, Y. *A Border Gateway Protocol 4 (BGP-4)*, Mar. 1995. RFC-1771.
- [31] TRAINA, P. *BGP-4 Protocol Analysis*, Mar. 1995. RFC-1774.
- [32] UC BERKELEY. The Millennium Project. <http://www.millennium.berkeley.edu>.

- [33] VILLAMIZAR, C., CHANDRA, R., AND GOVINDAN, R. *BGP Route Flap Dampening*, Nov. 1998. RFC-2439.
- [34] WAITZMAN, D., PARTRIDGE, C., AND DEERING, S. *Distance Vector Multicast Routing Protocol (DVMRP)*, Nov. 1988. RFC-1075.