Quilt: A Patchwork of Multicast Regions

Qi Huang[1,2], Ymir Vigfusson[3], Ken Birman[2], Haoyuan Li[2]
[1]School of Computer Science and Technology, Huazhong University of Science and Technology
[2]Department of Computer Science, Cornell University
[3]IBM Haifa Research Lab
qihuang@mail.hust.edu.cn, {ken, haoyuan@cs.cornell.edu}, ymirv@il.ibm.com

ABSTRACT

Network bottlenecks, firewalls, restrictions on IP Multicast availability and administrative policies have long prevented the use of multicast even where the fit seems obvious. The confusion around multicast poses a problem for large-scale pub/sub-based applications that need blazing speed even across WAN networks. There are a number of multicast protocols, but none is universally available. Thus relatively few applications are able to exploit multicast technology. Here, we present Quilt, a system that automatically weaves a patchwork of multicast regions each running different protocols, creating an efficient and scalable wide-area overlay. By dynamically exploring the environment at and between end-hosts, Quilt clusters nodes into patches, selecting the best multicast protocol from a developer-provided set on a patch-by-patch basis and adapting as needed. Quilt orchestrates inter-patch forwarding to maximize reliability while minimizing duplication. This paper discusses and then evaluates the system. We find that Quilt is an effective, backwards compatible option for supporting multicast wide-area networks.

1. INTRODUCTION

Publish/subscribe [14] has been a popular paradigm for building event-driven dissemination services and has found applications in diverse areas such as web management [39], social networks [15] and massively multiplayer online games (MMOGs) [5]. A natural transport mechanism for publish/subscribe communication is *multicast*, providing one-to-many message dissemination both efficiently and scalably. A standard wide-area multicast service has been on the wish-list for decades – a universal service that can distribute content over the increasingly complicated network while minimizing latency and network overhead. But, as we will explain below, no existing technology addresses the full range of issues confronted in real deployments. Among the culprits are restrictions on network-level multicast (IP multicast) [35], heterogeneity of Internet hosts, or problems with the specific approach such as large overhead or high latency outside of local area networks. With users at the edge of the cloud eager to adopt IP radio and IPTV [33, 36] and to experience network-enabled games and social platforms, and with cloud providers scaling out and providing global services between geographically distributed data centers, a multicast protocol that could be used safely throughout the Internet is an appealing goal.

To formalize the issues, consider an application that disseminates information from a single source to a set of peers. From the application perspective, a multicast service should satisfy several objectives:

1. It should minimize redundant network traffic on bottleneck links, routers and end-hosts.
2. It should minimize the mean latency of delivery while achieving the required throughput.
3. It should require limited per-node storage.
4. It should be robust to node churn/failure[1].
5. It should automatically adapt to the runtime environment.

There are a tremendous variety of multicast protocols [13, 16, 2, 9, 11], each specialized to different needs and optimized for different runtime conditions and technology options [35]. Nonetheless, each of these options has problems when evaluated against our goals.

Network-layer multicast, specifically IP Multicast (IPMC) [13], is widely supported in hardware and software. IPMC-enabled routers automatically build spanning trees for content distribution multicast groups and include mechanisms for robustness [16, 27], satisfying the first four goals. The abstraction for end-hosts is simple: any node can join, leave or send packets to a multicast group by using a dedicated IP address assigned to the group. But IPMC is rarely enabled across global providers in the Internet WAN, and only sometimes available within data centers or enterprises [35]. Reasons include concerns that the technology could be costly or disruptive, as well as economic and security issues that are unlikely to be solved anytime soon.

Application-layer multicast. At the other end of the spectrum one finds a plethora of application-level multicast (ALM) protocols, in which multicast is handled as an end-to-end mechanism running in the application itself. The simplest ALMs do the obvious, connecting senders directly to receivers, and then implementing multicast as a series of

[1]We do not assume multicast reliability or flow control, but do expect the multicast infrastructure to handle joins/leaves/crashes.

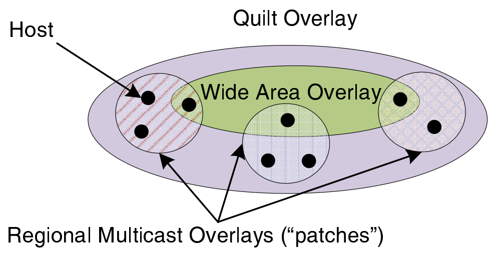


Figure 1: *Quilt automates the construction of complex multicast overlays. In this example, the system sews together a collection of regional multicast overlays to create a single virtual meta-overlay, separating the multicast interface from the implementation. Quilt permits the simultaneous use of various multicast solutions, each specialized for different runtime conditions.*

point-to-point unicasts. While easy to build, such schemes fall short of the first three goals in our list.

More sophisticated ALMs, such as NICE [2], SCRIBE [9] and ESM [11] organize end-hosts into dissemination overlays composed of unicast connections between pairs of nodes, again using TCP or UDP. However, few ALMs shape themselves to match the underlying network topology. As a result, information may traverse bottleneck links or long WAN paths multiple times, wasting bandwidth and increasing latency. Additionally, ALMs rarely leverage IPMC. If IPMC is available, it can be a good option within data centers. Indeed, failing to leverage IPMC results in unnecessary traffic that burdens switches and routers, increases sender load, and may increase latency.

Many ALMs, such as the ones listed above, have an underlying peer-to-peer (P2P) structure. Studies have shown that P2P overlays are vulnerable to node failures and churn: the neighbor sets can become ineffective due to unreachable or dead peers [23, 24]. The presence of inaccessible neighbors can disconnect a multicast tree, hence some P2P ALMs (e.g., DONet [41], Bullet [25] and Chainsaw [32]) organize end-hosts into a mesh-structure, increasing robustness, but at the cost of increased latency and overhead.

Combined protocols. Because the conditions for multicast are different inside and outside of a data center, one could also just deploy multiple solutions, side by side. In this approach, which Quilt adopts and generalizes, an application spanning data centers and WAN will need to combine two or more multicast infrastructures. At each participating node, this entails selecting the option best matched to local conditions. To combine the resulting disjoint overlays into a single overarching structure, certain nodes would then be selected to forward between the overlays where they overlap.

The concept of combining protocols has been explored in some prior work, but never in a realistically complex network. For example, WiMA [22] reduces streaming bandwidth consumption in wireless environments by deploying regional IPMC agents. Plumtree [26] and mTreebone [38] connect hosts to both tree and mesh overlays to accelerate content delivery while resisting churn, but the solution brings significant system overhead and latency. In SplitStream [10], nodes are organized into multiple tree overlays to create disjoint data paths for robustness. The resulting structure, however, is ultimately equivalent to the mesh suggested by [29] and neither is ideal. Here, we seek a solution that satisfies all the requirements listed earlier.

2. QUILT

Quilt weaves a collection of multicast regions into a "patchwork overlay", routing messages between overlays to optimize various objectives (Figure 1). The system operates as a library. The developer of an application registers the multicast options and provides a collection of executable selection rules that embody preferences and other logic. Network administrators impose policy, for example by enabling IPMC only on certain subnets. Quilt can then discover the context in which each end-host is running using automated tests, put the end-host in contact with regional peers, and finally optimize to create a patchwork and a forwarding graph that will achieve efficiency and robustness with minimal packet duplication.

Quilt innovates in several ways. The system incorporates a tool that runs on each node to discover its connectivity options, network topology and performance characteristics of upstream and local network links. This information is gathered using the kinds of user-level mechanisms employed in network tomography solutions. Quilt then runs a decentralized protocol that forms patches from nodes that are close with respect to latency and have similar network properties. For example, nodes residing in an IPMC-enabled data center are likely to belong to the same patch.

Within each patch, Quilt runs the "inner" protocol best matched to the properties of the environment. To sew patches together, Quilt builds a spanning global patch. It then identifies overlap regions, and selects forwarding nodes in a bottleneck-aware manner, but with sufficient redundancy to avoid disconnection in the event of churn. The global patch runs a protocol structured around latency-aware *OMNI trees* [3], which are regional spanning trees constructed to minimize average root-to-node latency, but with constrained outdegree for interior nodes (see Section 3.2.3). The following two use cases motivate these choices, and will also form the basis of our evaluation:

Real-time event dissemination. As mentioned before, data center hosted pub/sub services like Facebook, Twitter or IBM WebSphere need to disseminate messages, such as status updates of a web application or a human user, to groups of subscribers in real-time. All these systems use client-server structures: end-user messages or actions are relayed into data centers, where processing is performed, and then messages are disseminated both within the data center itself and to group members who should see the resulting updates. The data center itself may be broken into multiple geographically distributed regions. In this set-up, we see three kinds of regions: within a single data center, between centers, and from data centers to listening end-hosts. Quilt could use IPMC (or a fast ALM overlay) within each data center where the protocol is enabled, and a low-latency OMNI tree for message dissemination between data centers and to the end-users.

Internet broadcast. Consider an edge-oriented pub/sub style multicast application, such as the broadcast of a physical event or a MMOG with players situated on a university campus or in dorms. To the extent that data center hosts participate, there may not be much value in treating them differently from other kinds of high-capacity end-hosts. This perspective favors purely peer-to-peer protocols like the BitTorrent [20] family of content distribution overlays. Yet even here one may wish to treat the system as heterogeneous. In particular, Quilt can leverage efficient intra-patch protocols within campus networks, regional cable networks or an autonomous system (AS) where multiple interested users reside. However, rather than treating all users as endpoints in a single wide-area overlay, Quilt automatically groups users within a single region. This allows the broadcast source or the MMOG to avoid redundant transmission over the expensive or congested links that traverse firewalls or bottlenecks and are used to interconnect different domains. The patches are then linked with a wide-area OMNI tree.

Roadmap. The remainder of this paper elaborates on the environment-detection capabilities (Section 3) and overall design of Quilt (Section 4). We then explore the performance of the system under a variety of conditions running the real code on a simulated network (Section 5). The test scenarios are inspired by the event dissemination and Internet broadcast examples above. All network topologies used are from real platforms, and our code can be used, unchanged, in diverse environments.

3. ENVIRONMENT METHODOLOGY

In Quilt, the formation of multicast patches is driven by an environmental sensing mechanism. The mechanism runs in three stages. First, each Quilt end-host generates a tuple of environment information that represents the setting in which it was launched. An extensible set of tests are used to detect this information, and we describe the main ones below. Next, the environmental tuples are collected and aggregated to decide which patches will be created and which nodes belong to each. One of these will be Quilt's spanning "global" patch: just another patch in the terms of our framework, but designed to function as an overlay optimized for low delivery latency that also provides redundancy and low duplication rates. Finally, Quilt configures the end-host nodes to relay multicasts from patch to patch in accordance with the computed routing structure.

3.1 Environment Information

We use the term Environment Unique Identifier (EUID) to denote the data structure representing environmental information discovered by Quilt. Quilt's EUID is flexible, and can represent both individual end-hosts or entire multicast patches. In the former case, the EUID is associated with a particular host's unique identifier (UID) and the values in the tuple are the ones measured by the Quilt library when the host was launched, and then updated as it ran. If a single host has multiple network interfaces with different IP addresses, each NIC will have its own EUID[2], since different networks often have different properties. In the latter case, the EUID is associated with a UID designating the multicast region, and the associated values summarize the collection of end-hosts that Quilt aggregated into that region.

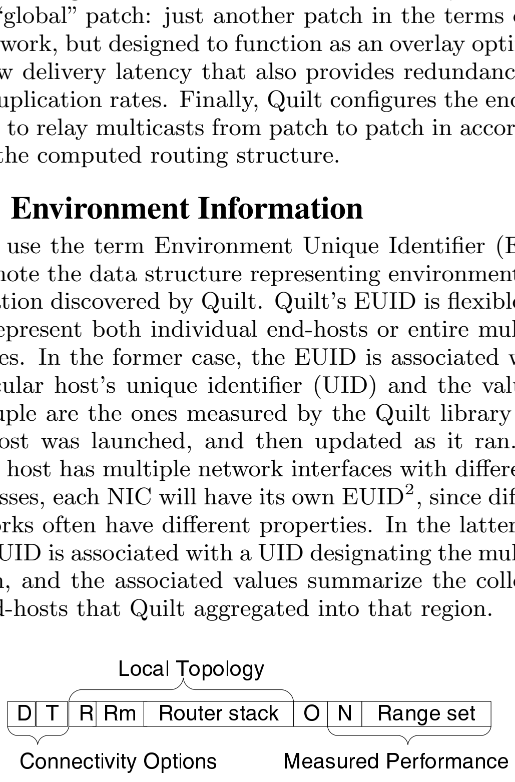


Figure 2: *Components of the Environment Unique Identifier (EUID).*

[2]Here we assume only one IP address per NIC, and therefore associate EUIDs with NICs.

As shown in Figure 2, each EUID includes three categories of information: *connectivity options*, *local topology* and *measured performance*.

Connectivity Options capture host accessibility by transport protocols such as TCP and UDP. Several research measurements [8, 4, 18] have found that over 74% of Internet hosts run behind NAT boxes and firewalls, and that connectivity barriers are important concerns for the ALM overlay construction [18]. For example, hosts isolated behind a NAT or a firewall may be limited to a leaf role, downstream from some other host running in a less constrained environment. As the name suggests, a connectivity options tuple (D: Direction, T: Transport protocol) indicates whether a given host or NIC can be reached using TCP or UDP, and also whether or not a NAT or firewall barrier is present on the path. Some hosts have multiple NICs and there are cases in which a single host has a mixture of bidirectional and unidirectional capabilities even on a single NIC. Such cases are handled using multiple tuples; Quilt will favor the "better" tuple entry in any set under a developer-controlled prioritization policy, as discussed further in Section 4 when we describe the ALM components and the roles of the Quilt bootstrap service.

As we will see below, the Quilt EUID plays a role that generalizes "system membership" information in standard settings. Whenever a Quilt component learns about some pair of end-hosts, it also learns about the connectivity tuples advertised by each, and hence will also be able to make a sensible estimate about the feasibility of linking them. EUIDs are used both in the (centralized) Quilt bootstrap service and in its ALM components, which run directly the end-host platforms. Our approach assumes that if a node advertises connectivity (and hasn't crashed), other nodes will be able to establish connections in the manner indicated. The scheme is tolerant of some level of mistakes, but errors can harm the quality of protocols that depend on connectivity estimation. Accordingly, Quilt employs conservative tests to create these constraint tuples.

Local Topology is a data structure characterizing the routing path from an end node to the Internet WAN. Because the Internet structure is tree-like [34], physically nearby hosts have a high likelihood of sharing routers on the path to the Internet core. Enterprise load balancing technology and "fat-tree" designs [31] can result in network topologies that violate this assumption. However, those mechanisms have limited impact near the edge. Our experiments show that it suffices for Quilt to interrogate four hops towards the Internet core.

Our main objective is to detect shared paths, for several reasons. First, this helps Quilt form locality-aware patches that optimize with respect to per-patch performance objectives. Additionally, local topology captures IPMC availability within regions by detecting the routers along the path. If Quilt reports that two hosts reside behind the same firewall or NAT, it can be deduced that those hosts should be able to communicate with one-another in a bidirectional manner, using the protocol specified in the EUID.

Quilt creates the EUID by traversing the first few network hops in the direction of the Internet core, using the local DNS server as its target (see Section 4.4). It then forms a "router stack" describing the path, consisting of the distance in hops to each router R or IPMC enabled router Rm. Techniques to discover local DNS servers and interme-

Table 1: *Multicast rule interface (A is a sample multicast protocol)*

Procedure	**Usage**
`bool IsPermitted(`n`)`	True iff node n with EUID $n.e$ is permitted to run protocol A
`bool CanJoin(`n`, `p`)`	True iff node n with EUID $n.e$ can join a patch p with EUID $p.e$ using protocol A
`void Join(`n`, `p`)`	Add member n with EUID $n.e$ to patch p with EUID $p.e$
`EUID CreatePatch(`n`)`	Create a new patch p consisting of node n, returning the EUID $p.e$

diate routers have been used in systems used for Internet geolocation [40], but additional overlays are required to calibrate the geographic position in those cases. Although we believe our specific technique is novel, it was motivated by a 2002 AT&T study [30], which determined that 64% of Internet hosts and their local DNS server belong to the same AS, and 75% of these DNS-host associations are within 5 hops of a common router on the `traceroute` path. This means that with at least 85% probability, hosts that share the same target DNS server or an intermediate router within 5 hops will share the same AS number. Based on later measurements [8], the probability is at least 80%.

Measured Performance estimates network performance. For a single host, we simply measure bandwidth and latency, then re-measure these values periodically. Performance metrics for a multicast region are computed as an aggregate over the individual values, including additional data representative of peer-to-peer performance. Because bandwidth and latency fluctuate, Quilt represents these metrics as *ranges*. Performance testing can be expensive and isn't always necessary, hence Quilt offers this service but will only employ it as requested by the ALM protocols that the developer selected. The performance testing component of Quilt is designed to be extensible, and we expect that down the road, individual ALMs might come with their own performance testing suites. The O bit signals the presence of an vector N of performance measurements in the EUID, each represented as a triple: (type, lower bound, upper bound).

3.2 Multicast Environment Rule

All Quilt entities (end-hosts using Quilt, as well as the bootstrap service) share a library of ALM protocols provided by the application designer. However, only some of these ALMs will be suitable in any given setting. Quilt's job is to only activate those protocols for which the runtime environment is appropriate. Accordingly, each protocol comes with an executable "rule" that Quilt can apply to a set of EUIDs to determine which ALMs are capable of running on the corresponding hosts. Abstractly, this kind of testing can occur anywhere; in practice, the most important use of the multicast rule set turns out to be on Quilt's bootstrap service.

Rules operate on sets of EUIDs, which can describe either individual end-hosts or entire multicast patches; in the latter case, the single EUID is understood to apply to all end-hosts belonging to the multicast patch. The rules themselves can embody arbitrary developer-supplied "intelligence", although the ones used in our prototype are fairly simple. A typical rule will be described in Section 3.2.1; it has the role of grouping end-hosts that can all communicate over a single IPMC address into an IPMC patch.

The job of Quilt on a newly launched end-host is thus as follows.

- First, it generates an EUID for each NIC associated with the end-host.
- Quilt then uploads this information to the bootstrap service, which responds with information about the ALMs to initialize, the peers that each of those ALMs should use as initial contacts in their multicast patches, and the EUIDs for those peers and patches. Recall that in Quilt, any host that learns of another host or of a multicast patch simultaneously learns its EUID.
- The ALMs then take over, running whatever protocol they like - perhaps peer-to-peer; perhaps using additional ALM-specific servers. The ones we use here run in a purely peer-to-peer mode once initialized, but actively manage themselves, repair themselves, and support dynamic membership changes (joins, leaves and failure handling).

Rules play a key role in multicast patch formation. The construction is done by comparing each node's EUID to the EUID of existing patches. All operations on a specific rule use the standard interface shown in Table 1 with which the aforementioned developer-supplied "intelligence" can be expressed. Given a collection of candidate patch members, each ALM-specific environment rule determines a maximal subset (patch) on which that ALM can run. Note that the subsets computed by different ALMs may overlap because they are determined independently. The candidate region is then covered with multiple multicast patches according to administrator specified priorities on the ALM environment rules. The administrator should define at least one rule for inter-patch multicast to ensure that the patches may be sewn together. Moreover, the administrator should consider enabling additional protocols to allow the application to continue functioning should network-level multicast suddenly become disabled.

We now illustrate the rules employed by our prototype ALMs: pure IP Multicast, DONet and an OMNI tree ALM which can be used either as a regional protocol or the tunneling protocol. Section 4.5 then explains how distinct multicast patches are "sewn together".

3.2.1 Rules for IP Multicast

The IPMC ALM simply creates a socket bound to a fixed class-D IP Multicast address and implements multicast send and receive using the standard system calls. The associated rule needs to ensure that if two nodes activate this ALM in the same multicast patch, they will be able to exchange IPMC datagrams.

The rule forms patches by testing EUID pairs. Let $E_i.S$ denote the set of IPMC enabled routers in the router stack in EUID E_i. Note that the cardinality of $|E_i.S|$ equals $E_i.R_m$. Given two hosts with EUID values E_1 and E_2, Quilt can first check whether each of them is connected to multicast enabled routers by verifying that $E_1.R_m > 0$ and $E_2.R_m > 0$. Quilt must then confirm that the hosts can connect to a single IPMC tree, that is $E_1.S \cap E_2.S \neq \varnothing$. In this case, the hosts share a multicast enabled environment and hence can be grouped into a single IP Multicast patch.

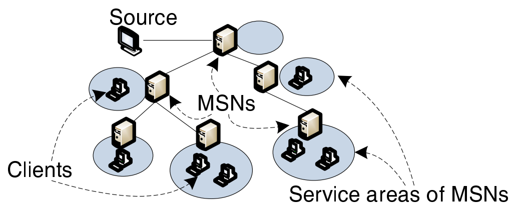


Figure 3: OMNI Tree: *An ALM overlay over Multicast Service Nodes (MSN) built to minimize the average source to client latency.*

If successful, we assign the patch a new merged EUID E_{12} with multicast-enabled router set $E_{12}.S = E_1.S \cup E_2.S$ and $E_{12}.R_m = |E_1.S \cup E_2.S|$. In the case of a third joining host, we create EUID E_{123} in the same manner, and discard EUID E_{12}. When the process terminates, the members of a given multicast region will all hold a single, maximal EUID for that region. For simplicity, the regional EUID is not updated if an end-host departs as such a host may reappear later, hence it changes only if new end-hosts later join. As noted earlier, the hosts excluded from one IPMC patch may still be able to form some other IPMC patch, for example if there are two multicast regions separated by a router that blocks multicasts.

3.2.2 Rules for DONet

For end-hosts running on the "edge" and lacking multicast support, for instance DSL cable users, Quilt supports the Data-driven Overlay Network (DONet), an ALM that constructs a latency-biased randomized mesh-structured overlay. DONet uses TCP as its link protocol and then uses a BitTorrent-based protocol to distribute multicasts in much the same way that content distribution networks share content. The DONet selection rule must ensure that within any multicast region, all end-hosts can form TCP links to some other end-hosts also in the region, but there is also an effort to ensure that peer-to-peer latencies for these links are roughly "comparable". Latency bias is then implemented by grouping hosts in a locality-aware fashion. Accordingly, define $C_{E_2}^{E_1}$ as the set of contacts from EUID E_1 to E_2. Then we say E_1 and E_2 are *compatible* if $C_{E_2}^{E_1} \cap C_{E_1}^{E_2} \neq \varnothing$. If EUID E_1 of some host in a DONet patch P is compatible with EUID E_2 of a joining host, then the new host should join patch P. If multiple patches are available, a host joins the patch which minimizes the predicted link latencies.

3.2.3 Rules for OMNI trees

As shown in Figure 3, the OMNI tree is a backbone-based ALM overlay built among Multicast Service Nodes (MSN), which are actually service proxies for normal clients [3]. Assuming that clients are close to their proxies and thus that the local latency can be neglected, the OMNI tree optimizes the average root-to-client latency based on the root-to-MSN latency and the number of clients each MSN is serving, but with constrained outdegree between MSNs. The protocol can run over both TCP and UDP, and provides low latency for all clients. Thus OMNI trees can be used as a regional multicast forwarding structure in situations where nodes need to be interconnected in a manner that minimizes latency.

Quilt also uses the OMNI tree to sew patches together, obtaining a reliable composite multicast solution in which packets follow low-latency routes. In this mode, the OMNI tree formation rule selects the subset of nodes on which to construct the tree, picking just a few representatives from each multicast patch for use in the tree (as MSNs). In selecting this subset from each patch, we try to favor well-endowed nodes and to minimize latency, but must also avoid the risk of disconnection: we select k end-hosts to relay messages so as to obtain tolerance to $k-1$ faults, unless (of course) there are fewer than k candidates available.

The OMNI ALM rule is that the new joining host must be able to contact some existing host. In other words, a new host E_2 can join the OMNI tree patch P if there exists a host E_1 in P with $C_{E_1}^{E_2} \neq \varnothing$. Our OMNI tree construction procedure selects k members of an overlap region to perform relaying, picking them in the order they joined the system, but with a further constraint: if there are more than k candidates, Measured Performance is used to pick the highest capacity end-hosts.

Although the OMNI ALM can dynamically heal and regenerate a damaged tree, fault-tolerance issues may arise when $k = 1$ since patches can get disconnected from the tree when representatives leave. To address the issue, Quilt stays in touch with the bootstrap service, which has a bigger picture of the network. Should a representative leave its patch, the service will select a new representative to replace the lost one, and the OMNI tree will then restructure itself "as ordered", restoring full functionality. Moreover, when $k > 1$, we run the risk that each multicast can be forwarded k times between different patches. This is where a duplicate suppression mechanism kicks in. The mechanism seeks to reduce the frequency of multicast duplication so that it won't represent a significant source of overhead. Sections 4.7 and 4.8 will talk about these mechanisms.

4. SYSTEM DESIGN

We now describe the Quilt architecture in more detail (Figure 4).

4.1 Architecture

Quilt provides a simple and standard API to the end-user application. Applications can treat Quilt as a black-box multicast service, and indeed our prototype applications do so. In the future we envision more sophisticated applications that might need to control the choice between multicast candidates using criteria beyond the ones supported by Quilt.

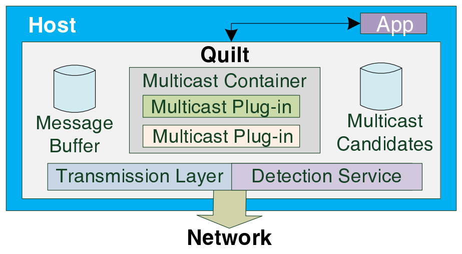


Figure 4: Quilt architecture. *The application treats Quilt as a single multicast overlay, but the internal structure of the system is more like an operating system supporting multiple plug-in "drivers". In our case, these drivers are multicast protocols, each specialized for some set of conditions. Our prototype includes three such protocols: the OMNI tree, an IPMC overlay, and an ALM called DONet.*

The Quilt system itself is object-oriented, organized as a set of components that interact through well-defined interfaces. The system exploits the dynamic composition features of .NET, running "native" on Windows and under Mono on Linux. We have implemented Quilt and the experiments in Section 5 ran actual code on Windows. The source code has been made publicly available as part of Live Distributed Objects platform [12] from Cornell.

4.2 Bootstrap Service

As seen in Figure 5, Quilt's bootstrap service runs free-standing on an Internet-accessible platform and plays a variety of roles. Our prototype is accessed using a standard HTTP-based remote method invocation. It must have access to the ALM-specific selection rules discussed earlier.

The bootstrap server has several roles. The first is to maintain a database with partial membership for each patch. The emphasis here is on partial: Rather than insisting on accuracy at potentially high cost, we track just a few (tens) long-lived members. The second role is to check multicast rules for newly joined nodes, thus structure them into patches. The third role is to maintain the healthiness of "Inter-Patch", keeping regional patches sewn together. Detailed processes of how bootstrap server works are described in later sections.

Quilt's bootstrap service design is centralized, but off the critical path, taking actions only when nodes join or if a patch becomes isolated as a result of node departures. The algorithms used are cheap (linear in the number of patches). We tested with up to 1000 nodes, and believe that far larger numbers could be supported.

Although the current Quilt bootstrap service is a single point of failure for the system, the code is deterministic and could easily be replicated for fault-tolerance, for example by replicating the servers to form a virtually synchronous group [6]. Much as a modern data center like Amazon.com routes requests to a set of front-end nodes, we can treat such a group as a single entity "from the outside" even as its members cooperate to respond to requests in a fault-tolerant manner. The virtual synchrony model permits parallel processing of requests, hence the work of patch formation could be load-balanced over the set. In larger settings one would generalize this approach, using a hierarchy of bootstrap servers, much like the hierarchy of DNS servers. Each network domain would connect to the appropriate server group. In this case, the failure of a regional server would impact the portion of the network it handles, but the remainder of the Quilt overlay would remain live.

4.3 Multicast Container

The Quilt Multicast Container can be understood as a storage structure for active protocol "objects". Quilt compatible ALM protocols implement a standard interface, which includes methods to initialize the protocol, to join a group (Quilt supplies an initial membership list), to send multicasts, and to deliver incoming messages to pre-registered handlers. Each multicast patch will span a set of end-hosts, and can maintain its own peer set in whatever manner the protocol dictates.

As protocols are selected, they are dynamically instantiated, a process roughly analogous to plug-in driver activation in an operating system or object creation in Java/C#. Multicast instances share the transmission layer and message buffer, which includes a window of recently received multicast messages. Use of these features is optional and protocol-specific, but can substantially simplify protocol logic.

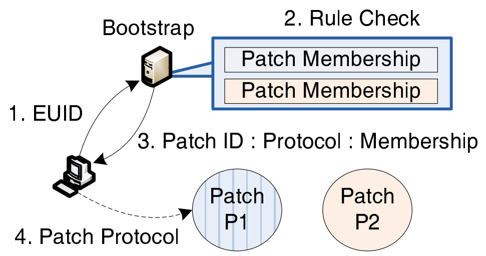


Figure 5: Patch formation. *Each joining host sends its locally produced EUID to the bootstrap server, which in turn finds an appropriate multicast patch (P1) for the host using various multicast rules and then returns the patch ID, protocol name and the initial membership.*

4.4 Detection Service

As discussed in Section 3, when Quilt is first launched on a node it needs to construct the set of EUIDs for that node's NICs. Details concerning the implementation of these mechanisms are omitted for brevity; we refer the interested reader to our technical report [21]. Briefly, connectivity options are detected using a modified version of NAT Checker [17]. This tool discovers upstream NAT boxes and tests to see if the NAT can be traversed. We extended it to check for firewalls by testing connectivity through a target server. The local router stack is discovered by running `traceroute`[3] to the local DNS server over the relevant NIC. Performance metrics such as latency and bandwidth are collected as part of the traceroute operation.

4.5 Patch Formation

As seen in Figure 5, once its EUID is computed, a new end-host sends a join message to the bootstrap server with its EUID value. The server applies the environment rules to check whether the new host can be added to some existing multicast patch. Suppose that our new host matches the patch P1. The bootstrap server returns contact information for the patch (partial membership that the ALM can use to peer with other P1 members), along with its patch ID and protocol name. With this information, the joining host checks its Multicast Candidates, finds the matching protocol plug-in and loads it into its Multicast Container component. The host launches the matching multicast protocol plug-in and supplies the initial membership data.

Now the new host can contact its peers within the patch P1, join the group and conduct operations defined inside the specific protocol. If a joining host matches several existing patches, the same process will be used, and multiple ALMs will be initialized. There are also cases in which a single ALM would be initialized more than once. For example, a single host might overlap with two IPMC patches, each using a distinct class-D IPMC address; it would then run the IPMC ALM once for each patch.

4.6 Inter-Patch Formation and Maintenance

As we learned earlier, a Quilt overlay consists of an OMNI

[3]In cases ICMP or UDP is unable to perform the detection, there are also other options like `tcptraceroute`[37].

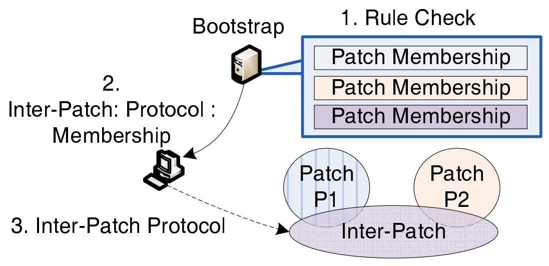


Figure 6: Inter-patch connectivity. *New nodes are assigned to join a regional multicast patch by the Quilt bootstrap service, along with an inter-patch multicast protocol if the patch does not have enough representatives to link to other patches.*

"inter-patch" ALM tree that connects regional patches running other ALMs, overlapping at k representatives. The bootstrap service constructs and then maintains this tree.

The bootstrap service selects inter-patch representatives, instructs them to play the patch-to-patch tunneling role, and then monitors their health. Should a tunnel node crash, the bootstrap selects some other tunnel node to take over the forwarding role. As shown in Figure 6, the OMNI inter-patch ALM is actually treated much like any other patch, although it uses a specialized patch construction rule, designed to ensure connectivity of the Quilt overlay as a whole, acyclic paths and robustness.

For example, suppose an application finds itself in the patch P1, but not the patch P2. Now a multicast occurs in the patch P2. How would it reach the new node? As seen in Figure 6, at least some nodes in the P1 will be assigned tunneling roles between the P1, the global OMNI tree that connects P1 representatives to P2 representatives, and the P2 ALM.

If our new node is selected to play this role, the bootstrap service will tell the node to initialize its OMNI inter-patch ALM in addition to the patch P1 ALM. The OMNI ALM will join existing OMNI inter-patch overlay nodes, extending the OMNI tree. Now, new P1 multicasts can be relayed into the OMNI ALM for delivery to P2 patch members, and vice versa. The forwarding rule can be understood as region specific: our P1 tunnel would forward "P1 multicasts" out via the OMNI ALM, and forward "P2 multicasts" in also via the OMNI ALM. A "P1 multicast" refers to multicast initiated by a source in the P1 region. The forwarding policy described here is cycle-free.

4.7 Churn Resilience

In Quilt, each ALM must implement its own mechanisms to handle node joins and departures. Quilt itself intervenes only when an ALM node that was playing a patch-to-patch tunnel role (a *representative*) departs. To give an extreme example, if all the representatives of patch P suddenly leave at the same time, A's ALM will lose connectivity to the OMNI tree.

Quilt has two mechanisms to resolve this issue. The first one is proactive and consists of designating k representatives (rather than just 1) from each patch. With $k = 2$ or $k = 3$, Quilt is much less likely to experience a global partitioning event, but of course larger values of k do increase the likelihood that a multicast will be sent more than once in the same ALM, a form of link stress. We measure this effect in our experiments.

Quilt's second robustness mechanism is focused on repair when its forwarding infrastructure is disrupted by a failure. In Quilt, each host periodically reports to the bootstrap server, allowing the bootstrap server to maintain a relatively "fresh" partial snapshot of membership for each patch. Representatives are monitored and if a failure is detected, dead representatives are removed. The bootstrap server will then select another host from its snapshot to become representatives and join the OMNI tree. With larger values of k, the reporting frequency can be decreased to reduce overhead. Further, normal hosts and representatives can report at different frequencies. Our experiments also suggest that with large enough k, system throughput will not be influenced by the frequency, which is currently set to tens of minutes. Moreover, a large enough value of k can also circumvent problems in the event that Quilt selects a failed node as a new patch representative. Accordingly, Quilt uses a range of values $\{k_{\min}, k_{\max}\}$. The bootstrap server steps in only when the current $k < k_{\min}$ at which point it picks $k_{\max} - k$ hosts to play patch representative roles.

4.8 Duplication Suppression

With more representatives, message duplication can happen both between patches (more contacts in remote patches) and inside patches (multiple representatives as patch sources). For ALM protocols, duplicate messages waste resources without necessarily improving reliability. Accordingly, representatives should not blindly forward messages. To help suppress duplicates, each tunneling host in Quilt maintains a Bloom filter [7]. With a few bitwise OR operations, a new multicast can be added to the filter; with a few bit tests, duplicates can be discovered (with some small risk of false positives). Since Bloom filters slowly fill and become ineffective, a new Bloom filter is initialized periodically, at a frequency tied to the multicast data rate[4]

Our experiments determined that while the Bloom filtering technique works well for most ALMs, applications that stream high data rates of multicasts into IPMC regions still experienced substantial levels of duplication. We thus developed a second duplication suppression mechanism specifically for this case. We use a gossip-based protocol to construct a small overlay linking just the patch representatives in a given region. Since k is tiny, these representatives find each other rapidly and can maintain a full all-to-all graph. We then run a simple 2-phase "synchronization" protocol whereby the representatives agree on which multicast each should forward. Our initial implementation is unsophisticated and yet, as seen in the experiments, turns out to be quite robust and imposes low overhead.

4.9 Multi-Source Support

Quilt combines multiple ALM protocols to construct a hybrid multicast overlay, hence support for multicast streams depends on the properties of the underlying ALM solutions: many scalable multicast protocols only permit a single sender. Quilt's DONet protocol can support multi-source messages with trivial changes, and its IPMC protocol needs no changes at all.

We distinguish several forms of multi-source applications.

[4]Each node populates a new Bloom filter with the multicast messages it buffered locally to reduce the rate of false negatives during initializing periods.

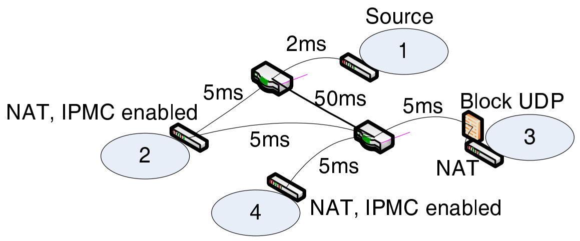


Figure 7: *Synthetic topology.*

In the first, a data center hosts some large number of servers, and each server is a distinct source of multicasts. If servers handle disjoint sets of clients, which is common in today's multi-user games, one could simply construct an independent Quilt overlay for each source.

In the second scenario, there are small numbers of multicast sources and they send to overlapping sets of clients. Here, the primary cost for Quilt involves the growth in size and frequency of the duplicate suppression messages used by our protocol, and the need to create a distinct Bloom filter for each source. If these costs are perceived as excessive (for example, if there would be tens or hundreds of senders to each Quilt client), the servers within any single data center could share a single source identifier and sequencer, much as a modern data center like Amazon.com uses network address translation to expose a small number of IP addresses to the outside world, despite using hundreds of thousands of servers to handle incoming load.

The most extreme scenarios involve large numbers of sources (for example, all-to-all broadcast within the Quilt client set). Such a pattern would be most easily handled by relaying the traffic through the Quilt bootstrap service, which could then number events sequentially and re-multicast them, functioning as a single source.

5. EVALUATION

5.1 Experiment Setup

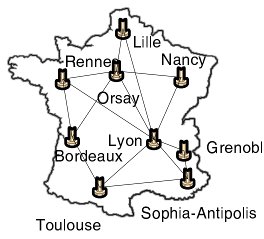
The remainder of the paper describes our experimental work. The goal is to show that Quilt achieves the objectives for a multicast service set out in the introduction, specifically that it minimizes redundant traffic, achieves low average latency, adapts to the environment and tolerates churn without imposing significant overhead. Although we created our own packet-level event-based simulator to simulate a variety of networks, accurately emulating the various topologies of interest, the simulator runs the Quilt code unchanged – it was implemented as a wrapper that virtualizes the system calls used by Quilt to interact with the outside world. Thus Quilt itself was not modified at all for experiments. For example, when we said earlier that `traceroute` is used to probe routers, that is precisely what occurs in our experiments. Only the network and its routing infrastructure is simulated. We focused on three topologies. The applications we evaluated in these networks both employ a single multicast source.

Synthetic Topology. Our first experiments explore multi-patch scenarios designed to exercise Quilt's tunneling functionality. We construct 4 domains, each with 5 hosts. As shown in Figure 7, with the exception of the multicast source domain, each domain gateway is a NAT. Domain 3 also has a firewall which blocks incoming UDP traffic. The gateway routers of domain 2 and 4 are IPMC enabled, while domains 1 and 3 lack IPMC support.

Table 2: *Multicast mechanisms evaluated.*

Mechanism	Synthetic	Data Center	Internet
Quilt ($k = 1$)	✓	✓	✓
Quilt ($k > 1$)		✓	✓
OMNI Tree		✓	
DONet			✓

Data Center Topology. Our second scenario stresses the real-event dissemination performance of Quilt by exploring an event notification (publish/subscribe) scenario among data centers linked on the Internet WAN. We based this topology on the published network structure of Grid5000 [19], a scientific computing infrastructure that links 25 clusters deployed among 9 sites over France, with 1531 servers in total (see figure on the right). As one would expect, the lowest latencies are for servers at the same site; for example, there is one cluster at Rennes with inter-node latency averaging 0.3ms, and the highest latencies within that site were 0.6ms. Servers at different sites experience latencies ranging from 4ms to 12ms. IP Multicast is enabled within each site but not between different sites.



Internet Topology. Our third scenario focuses on the wide-area Internet behavior of Quilt. PeerWise [28] provides end-to-end latency data for 1715 hosts, 951 of which have connectivity to one-another. We focused on those 951 hosts, and extracted the measured end-to-end latency information. To transform this latency data into an Internet test scenario, we then scanned some 400,000 `traceroute` records from CAIDA [1] during 2003 and 2004, and located 600 hosts that exhibited host-to-host latencies closely matched to the values from the PeerWise topology. For each host we collected the surrounding router path information. We then padded the set out to the full 951 hosts by duplicating settings for hosts that were in the same AS domain, using the PeerWise trace to identify them. IPMC is not available in this topology.

For each of the three topologies, we ran Quilt with a set of three multicast candidates: IP Multicast and DONet as intra-patch ALMs, and an OMNI tree for inter-patch communication. IPMC has obvious performance advantages, hence when more than one option is available, a host favors IPMC, then DONet, and only uses OMNI forwarding as a last resort. We treat TCP as our primary unicast protocol, employing UDP and UDP-based NAT traversal only if TCP is not available but UDP would work (a rare case). Table 2 shows the different mechanisms we evaluate in the three topologies.

5.2 Synthetic Scenario

We used the Synthetic Topology to evaluate the ability of Quilt to adaptively partition hosts into appropriate multicast patches in the manner of Sections 3 and 4. Additionally, because our topology includes various barriers (NATs, fire-

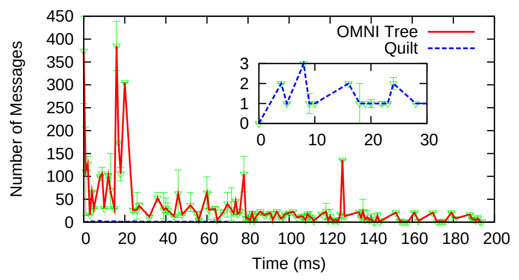


Figure 8: DATA CENTER*: Rate of message for constructing the tree overlay. The Quilt WAN overlay is small and inexpensive in comparison to a system-wide OMNI tree.*

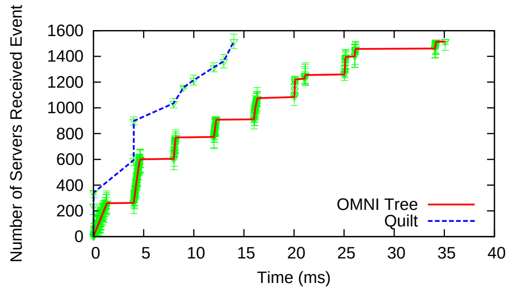
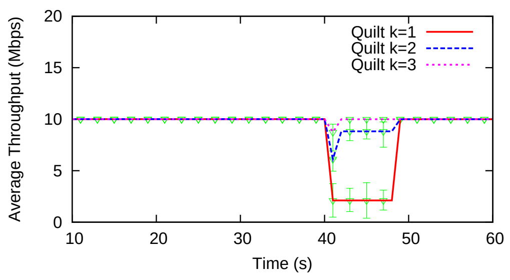
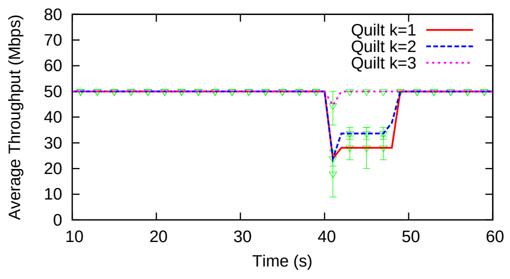


Figure 9: DATA CENTER*: Nodes running Quilt receive events faster over time compared to the system-wide OMNI tree because the system is able to leverage IPMC in regions where that option is available.*



(a) 10Mbps



(b) 50Mbps

Figure 10: DATA CENTER*: Average throughput while streaming at 10Mbps (a) and 50Mbps (b). Quilt recovers within about 10 seconds from a major disruption introduced at time 40; the recovery is faster with more representatives per region.*

walls, varied IP Multicast configurations), we used it to test our detection mechanisms in realistic conditions.

As shown in Table 3, hosts inside domain 2 and 4 form two patches, each using IPMC. Domains 1 and 3 construct DONet overlays. In our experiment, hosts joined in host-ID order, hence the lowest ID host in each domain tunnels for the inter-patch OMNI tree (here, $k = 1$). If latency were the sole metric, hosts 11 and 16 would be picked as children of host 6 in the tree. But host 11 is blocked by a NAT, and can only be connect to host 1 through a high-latency TCP link. Quilt configures this host to use the OMNI ALM. We conclude that Quilt correctly selects the appropriate ALM, and correctly links the resulting patches.

5.3 Data Center Scenario

Our second suite of experiments focuses on real-time event dissemination scenarios between a set of enterprises or data centers. Here we simulate the DATA CENTER topology, focusing on the quality of the overlay construction with respect to end-to-end multicast latency. Recall that Grid5000 permits the use of IPMC within (but not between) data center

Table 3: SYNTHETIC*: Message Delivery Summary.*

Domain	Host	Protocol	Method
1	1	DONet, OMNI	Source
	2∽5	DONet	TCP
2	6	IPMC, OMNI	TCP
	7∽10	IPMC	IPMC
3	11	DONet, OMNI	TCP
	12∽15	DONet	TCP
4	16	IPMC, OMNI	UDP tunnel
	17∽20	IPMC	IPMC

sites. We expect Quilt to discover and exploit the feature when possible, deploying a WAN OMNI tree that switches to IPMC within regional patches, thereby achieving a substantial performance benefit.

We compare Quilt with a simple, standard ALM: A pure OMNI overlay spanning the full node set. Such a solution runs entirely over TCP, with each message forwarded by tree nodes on each of their outgoing links.

Our simulation uses a randomly selected a server as its multicast source. This server joins the system, and then 1 second later, the other 1530 servers join simultaneously. We measure the time and overhead of constructing the OMNI tree and the latency in disseminating a multicast message to all the servers. Each result is the mean value of 10 runs.

Overhead. Figure 8 shows the traffic overhead and time required to construct a tree overlay in the two situations. Dealing with simultaneous join of 1530 hosts, the pure OMNI solution takes around 190ms to construct the overlay, and the traffic measured per millisecond ranges between several tens of messages to over 350 messages per second. In contrast, by assigning most of the hosts to regional multicast patches, Quilt is able to construct a much smaller OMNI tree for the case in which IPMC is available, consisting of just 8 nodes, each from a data center patch. This tree is formed in less than 30 milliseconds, and traffic is less than 3 messages per millisecond (embedded graph in Figure 8).

Latencies. Figure 9 shows the event delivery latencies we achieved in the two cases. With the pure OMNI tree ALM, it takes up to 40 milliseconds to multicast a single event to 1530 servers. The line grows in a "step by step" way due to the strictly hierarchical latency relationship among servers. When running Quilt, we can see the latency has been reduced. In roughly 14 milliseconds, all the servers

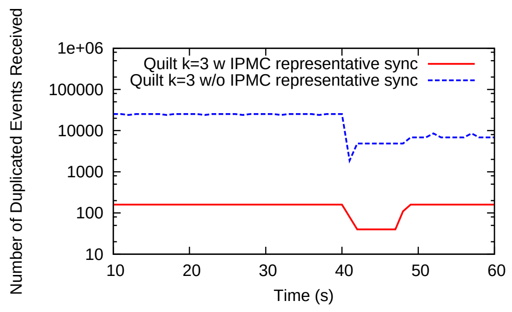


Figure 11: Data Center*: Rate of duplicate packets in our 10Mbps experiments with and without the "sync" protocol of Section 4.7. The mechanism effectively filters out duplicates for $k = 3$, the value which minimized throughput disruption in Figure 10.*

have received the multicast event.

There are two explanations for this performance gain. One is that in Quilt, each patch only contributes a single node to the inter-patch OMNI tree, decreasing the tree depth dramatically. Thus messages reach the sites more quickly. The second speedup reflects IPMC: Quilt routes multicasts from the OMNI network into the IPMC regions as they arrive in each site, and gives an additional latency improvement.

Churn Resilience. To evaluate the churn resilience of Quilt at various levels of throughput, we run Quilt with 10Mbps and 50Mbps multicast events. Quilt reaches a steady state within a few seconds. After 40 seconds, we introduce a catastrophic failure: 50% of the nodes, selected randomly, fail simultaneously. Figures 10(a) and 10(b) show the average throughput when experiencing the correlated failure. When the failure happens, the average throughput will drop dramatically in both figures for two reasons. First, in the OMNI tree, failed representatives at higher level would affect the throughput of representatives at lower level. Second, patches without responsive representatives cannot receive multicast events. In the experiments, the monitoring interval for representatives is set to 10 seconds. We see two effects. The first involves self-repair of the OMNI ALM: within about 2 seconds, the OMNI overlay is repaired, but to the extent that the crash disrupted the OMNI tree, some loss can occur. Next one sees a slower repair as Quilt's bootstrap server notices that patch representatives have failed and replaces them. As on would expect, with bigger k value, less loss occurs and indeed, when $k = 3$, all patches retain some responsive representatives.

Duplication Suppression. Next, we evaluated the performance of the algorithm from Section 4.7. Our experiment transmits a video stream consisting of a series of 1Mb video segments, transmitted at a data rate of 10Mb/s: 10 segments per second. We use k=3 (there are three forwarders into each region), and stress the overlay by injecting an abrupt churn event after 40 seconds: half the nodes in the system suddenly depart.

Our experiment treats each segment as a single Quilt multicast. The large segment size is deliberate: this amortizes the Quilt duplicate suppression mechanism over more bytes of data, so the effective overheads are reduced.

Recall that Quilt has two ways of inhibiting duplicates. The first is based purely on Bloom filters, and the second on an inexpensive 2-phase "sync" mechanism that runs among the representatives. We first looked at suppression using

Table 4: *Locality-aware partition performance*

Range	Partition#	Intra Latency	Inter Latency
3	513	23ms	473ms
4	403	37ms	518ms
5	327	127ms	973ms
6	241	131ms	1489ms

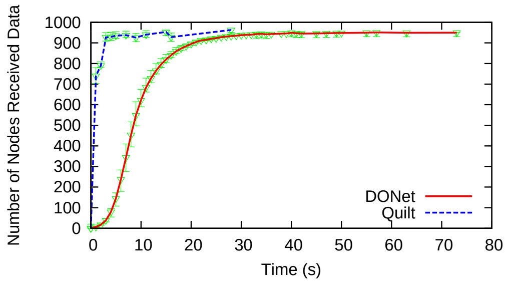


Figure 12: Internet*: Number of nodes receiving a single multicast as a function of time.*

the Bloom filter alone. Figure 11 shows the total number of duplicate segments received per second with k = 3. Each delivery of a segment to a single server is counted as a distinct "event" here, hence with 1531 receivers a perfect solution would yield 15,310 receive events per second and zero duplicates. An ineffective duplicate suppression scheme, for k=3, would deliver 15,310 legitimate receive events, but would have 30,620 duplicate receptions. As seen in Figure 11, Quilt's Bloom filter scheme roughly halves the worst-case duplication frequency for this particular mix of data size and data rate.

Next we enabled the 2-phase sync protocol. With this additional mechanism in use, Quilt pays a small cost in terms of latency. However the benefit is substantial: now the duplicate rate is a factor of 10 smaller, corresponding to a duplicate rate of about 1.2% at a typical receiver.

5.4 Internet Scenario

Our final test scenario examines a WAN Internet broadcast using the Internet Topology. Because IPMC is not generally supported on the public Internet, Quilt employs DONet as its intra-patch ALM. As discussed before, DONet implements a mesh-structured overlay, without explicit parent-child relationships. This mesh has multiple routes, increasing effective bandwidth. The DONet protocol [41] is similar in spirit to BitTorrent: nodes download needed data from peers, and offer multicasts retained in their multicast buffers to neighbors, sending on demand. Porting to Quilt involved minimal changes.

Recall from Section 3 that Quilt uses the router host to cluster hosts by latency. As a first step, we measured the effectiveness of this locality-aware grouping mechanism using the network topology generated from our PeerWise data set, augmented in the manner described above using CAIDA traces.

Table 4 shows node to node latencies within ("intra") and between ("inter") Quilt-generated partitions using EUID sets reflecting varying router-path lengths. We see that EUIDs measuring 4-hops towards the Internet core strike a balance: we obtain 403 small multicast patches with median intra-node latency under 40ms, and median inter-patch latency of

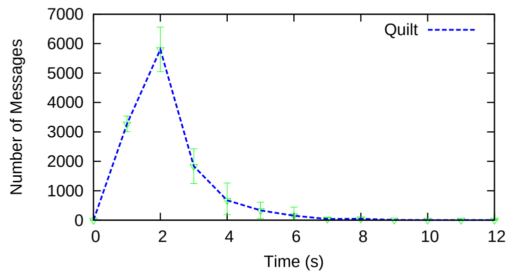


Figure 13: INTERNET*: Message rate while constructing the OMNI tree overlay for 951 hosts.*

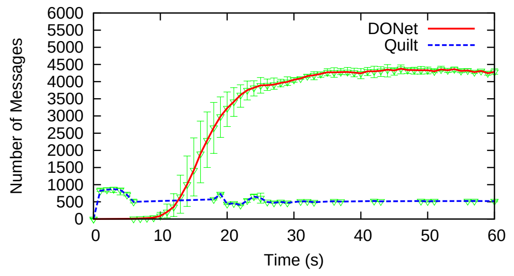
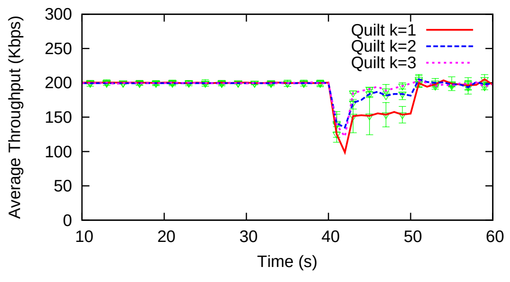
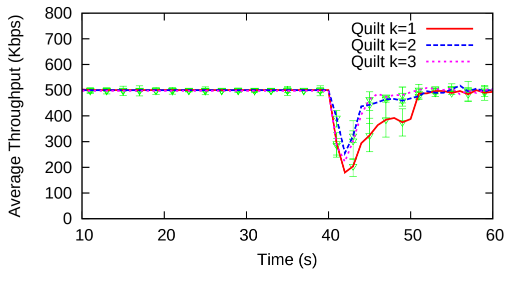


Figure 14: INTERNET*: Rate of control messages to send a single multicast data block.*



(a) 200Kbps



(b) 500Kbps

Figure 15: INTERNET*: Average throughput over time when sending 200Kbps (a) and 500Kbps (b) multicast streams. In this network, recovery is harder than in the one used for Figure 10 since many patches contain only one or two nodes.*

about 500ms. Quilt runs DONet within these patches and OMNI trees to sew them together.

Latencies. Figure 12 shows a CDF for delivery latencies for a single Internet broadcast message over the resulting structures. Quilt substantially outperforms the pure DONet solution, delivering the packet in just 4 seconds, compared to 60 seconds for DONet. The data is again averaged over 10 runs. The huge performance difference is easily explained: in DONet, as in any BitTorrent-based protocol, transfer of a message between two hosts requires three scheduling cycles: a so-called "buffermap" broadcast, a packet solicitation, and then a response containing the desired message. Each cycle requires 1 second in our simulation. If all hosts run DONet, with node degrees ranging from 4 to 6, a broadcast needs to traverse around 5 hops to reach the most distant receivers: an additional delay of about 15 seconds. Quilt's OMNI trees avoid the need for such a handshake.

Overhead. Quilt's lower latencies for delivery come at a cost as witnessed by the number of messages exchanged to construct the 951-node OMNI tree shown in Figure 13. DONet builds its mesh using an inexpensive random strategy. On the other hand, DONet pays such a high overhead to determine which packets to pull and request them that if we compare Quilt with DONet once the OMNI tree is in place, we see that Quilt takes the lead, with substantially lower control overhead (Figure 14). One explanation for this overhead reduction is that grouped patches have very few members due to PeerWise's sparse topology. More importantly, higher latency of data deliveries in pure DONet requires multiple scheduling cycles, resulting in additional control overhead for each of them.

Churn Resilience. In the INTERNET scenario, we also measure the system robustness under 200Kbps and 500Kbps traffic, picked to match the most popular streaming rates seen in the current Internet. As shown in Figures 15(a) and 15(b), the results are similar to the ones shown in Figures 10(a) and 10(b). There is an initial repair phase during which Quilt recovers its OMNI overlay tree, after which the value of k determines the delay of full recovery. With $k = 3$ recovery is quite rapid. Nonetheless, this experiment reveals two important differences from the data scenario used in Figures 10(a) and 10(b). First, the sparse PeerWise topology has many regions that are too small to provide k representatives. Thus even with $k = 2$, the system gains as much robustness as is available in these regions. From such an extreme failure (50% of the nodes), the overlay resulting from the crash is even sparser than it was in the first place. Additionally, the size of an OMNI tree can be quite different for $k = 2$ and $k = 3$. Thus, when catastrophic failure happens, a bigger tree can actually exhibit a lower average throughput in the worst case since there are more hosts in low levels of the OMNI tree. Our experiment shows that for all values of k, the OMNI tree has repaired itself within 10 seconds.

Duplication Suppression. For reasons of brevity, we have not graphed duplicate suppression data for this experiment. We found that for data rates below 500Kbps, the average number of duplicate events received per second is 50% lower for $k = 2$ and 63% lower for $k = 3$, or 86 and 114 before the correlated failure, and 22 and 56 after the failure.

6. CONCLUSION

Our paper starts with the premise that it is impractical to create one-size-fits-all multicast solutions for distributed applications that may include nodes separated by Internet WAN links, small clusters of nodes residing behind a shared NAT or firewall, and larger clusters running in settings where IPMC is available. Quilt addresses this need by creating a patchwork of multicast regions. Quilt senses the environment in which the client is running and automates the region formation process using a simple, extensible scheme. Quilt itself is lightweight and should scale easily to large deployments and to applications involving large numbers of multicast sources. The system is available for free download from our Cornell site in source form, and has been integrated with Cornell's Live Distributed Objects platform.

7. ACKNOWLEDGMENTS

We are grateful to our shepherd, Daniel Freedman, Chi Ho, Krzys Ostrowski and Robbert van Renesse for useful feedback, and to the Chinese National Science Foundation Project No.607-31160730, the National Science Foundation, the Air Force Research Laboratories at Rome NY, Intel and Cisco for their support of this research. The second author is partially supported by EU IST Project CoMiFin FP7-ICT-225407/2008.

8. REFERENCES

[1] Caida. http://www.caida.org.

[2] S. Banerjee, B. Bhattacharjee, and C. Kommareddy. Scalable application layer multicast. In *Proceedings of the SIGCOMM 2002*, Pittsburgh, PA, USA.

[3] S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller. Construction of an efficient overlay multicast infrastructure for real-time applications. In *Proceedings of INFOCOM 2003*, San Francisco, CA, USA.

[4] S. M. Bellovin. A Technique for Counting NATted Hosts. In *Proceedings of IMW'02*, Marseille, France, November 2002.

[5] A. R. Bharambe, S. G. Rao, and S. Seshan. Mercury: a scalable publish-subscribe system for internet games. In *Proceedings of NETGAMES 2002*, Braunschweig, Germany.

[6] K. Birman and T. Joseph. Exploiting virtual synchrony in distributed systems. In *Proceedings of SOSP '87*, New York, NY, USA, 1987.

[7] A. Z. Broder and M. Mitzenmacher. Survey: Network Applications of Bloom Filters: A Survey. *Internet Mathematics*, 1(4), 2003.

[8] M. Casado and M. J. Freedman. Peering through the Shroud: The Effect of Edge Opacity on IP-based Client Identification. In *Proceedings of NSDI'07*, Cambridge, MA, April 2007.

[9] M. Castro, P. Druschel, A. Kermarrec, and A. Rowstron. SCRIBE: A large-scale and decentralized application-level multicast infrastructure. *IEEE JSAC*, 20(8):1489–1499, 2002.

[10] M. Castro, P. Druschel, A.-M. Kermarrec, A. Nandi, A. I. T. Rowstron, and A. Singh. SplitStream: high-bandwidth multicast in cooperative environments. In *Proceedings of SOSP 2003*, Bolton Landing, NY, USA, October 19-22.

[11] Y.-H. Chu, S. G. Rao, and H. Zhang. In *Proceedings of SIGMETRICS 2000*, number 1, Santa Clara, CA, USA, June 17-21.

[12] CodePlex. http://liveobjects.codeplex.com/.

[13] S. E. Deering and D. R. Cheriton. *ACM Trans. Comput. Syst.*, (2):85–110, May.

[14] P. T. Eugster, P. Felber, R. Guerraoui, and A.-M. Kermarrec. The many faces of publish/subscribe. *ACM Comput. Surv.*, 35(2):114–131, 2003.

[15] Facebook. http://www.facebook.com/.

[16] S. Floyd, V. Jacobson, C.-G. Liu, S. Mccanne, and L. Zhang. A reliable multicast framework for light-weight sessions and application level framing. *IEEE/ACM TON*, 5(6):784–803, December 1997.

[17] B. Ford, D. Kegel, and P. Srisuresh. Peer-to-Peer Communication Across Network Address Translators. In *Proceedings of the USENIX'05*, 2005.

[18] A. Ganjam and H. Zhang. Connectivity Restrictions in Overlay Multicast. In *Proceedings of NOSSDAV'04*, Kinsale, County Cork, Ireland, June 2004.

[19] Grid5000. http://www.grid5000.fr/.

[20] L. Guo, S. Chen, Z. Xiao, E. Tan, X. Ding, and X. Zhang. Measurements, Analysis, and Modeling of BitTorrent-like Systems. In *Proceedings of IMC'05*, Berkeley, CA, USA, October 19-21 2005.

[21] Q. Huang and K. Birman. Solo: Self organizing live objects. Technical report, Cornell Univeristy, 2008.

[22] W. Jiang, X. Liao, H. Jin, and Z. Yuan. A Multicast Based Bandwidth Saving Approach for Wireless Live Streaming System. *International Journal of Smart Home*, 2(1), 2008.

[23] H. Kang, E. Chan-Tin, and N. Hopper. Why Kad Lookup Fail. In *Proceedings of P2P'09*, Seattle, WA, USA, September 9-11 2009.

[24] A.-M. Kermarrec, A. Pace, V. Quéma, and V. Schiavoni. NAT-resilient Gossip Peer Sampling. In *Proceedings of ICDCS 2009*, Montreal, Québec, Canada, June 22-26.

[25] D. Kostic, A. Rodriguez, J. R. Albrecht, and A. Vahdat. Bullet: high bandwidth data dissemination using an overlay mesh. In *Proceedings of SOSP 2003*, Bolton Landing, NY, USA, October 19-22.

[26] J. Leitão, J. Pereira, and L. Rodrigues. Epidemic Broadcast Trees. In *Proceedings of SRDS'07*, Beijing, China, Oct. 2007.

[27] J. C.-H. Lin and S. Paul. RMTP: A Reliable Multicast Transport Protocol. In *Proceedings of INFOCOM 1996*.

[28] C. Lumezanu, R. Baden, D. Levin, N. Spring, and B. Bhattacharjee. Symbiotic relationships in internet routing overlays. In *Proceedings of NSDI'09*, 2009.

[29] N. Magharei, R. Rejaie, and Y. Guo. Mesh or Multiple-Tree: A Comparative Study of Live P2P Streaming Approaches. In *Proceedings of INFOCOM 2007*, Anchorage, Alaska, USA, May 6-12.

[30] Z. M. Mao, C. D. Cranor, F. Douglis, M. Rabinovich, O. Spatscheck, and J. Wang. A Precise and Efficient Evaluation of the Proximity Between Web Clients and Their Local DNS Servers. In *Proceedings of USENIX 2002*, Monterey, California, USA, June 10-15.

[31] R. N. Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat. PortLand: a scalable fault-tolerant layer 2 data center network fabric. In *Proceedings of SIGCOMM 2008*, Barcelona, Spain, August 16-21.

[32] V. S. Pai, K. Kumar, K. Tamilmani, V. Sambamurthy, and A. E. Mohr. Chainsaw: Eliminating Trees from Overlay Multicast. In *Proceedings of IPTPS 2005*, Ithaca, NY, USA, February 24-25.

[33] PPLive. http://www.pplive.com/.

[34] V. Ramasubramanian, D. Malkhi, F. Kuhn, M. Balakrishnan, A. Gupta, and A. Akella. On the treeness of internet latency and bandwidth. In *Proceedings of SIGMETRICS 2009*, Seattle, WA, USA, June 15-19.

[35] S. Ratnasamy, A. Ermolinskiy, and S. Shenker. Revisiting IP multicast. In *Proceedings of SIGCOMM 2006*, Pisa, Italy, September 11-15.

[36] SopCast. http://www.sopcast.com/.

[37] tcptraceroute. http://michael.toren.net/code/tcptraceroute/.

[38] F. Wang, Y. Xiong, and J. Liu. mTreebone: A Hybrid Tree/Mesh Overlay for Application-Layer Live Video Multicast. In *Proceedings of ICDCS 2007*, Toronto, Ontario, Canada, June 25-29 2007.

[39] WebSphere. http://www-01.ibm.com/software/websphere/.

[40] B. Wong, I. Stoyanov, and E. G. Sirer. Octant: A Comprehensive Framework for the Geolocalization of Internet Hosts. In *Proceedings of NSDI'07*, Cambridge, Massachusetts, April 2007.

[41] X. Zhang, J. Liu, B. Li, and T.-S. P. Yum. CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming. In *Proceedings of INFOCOM 2005*, Miami, FL, USA, March 13-17.