Kevlar: A Flexible Infrastructure for Wide-area Collaborative Applications

Qi Huang[1,2], Daniel A. Freedman[2], Ymir Vigfusson[3],
Ken Birman[2], and Bo Peng[2]

[1] SCTS & CGCL, Huazhong University of Science and Technology, Wuhan, China
[2] Cornell University, Ithaca, New York, USA
[3] IBM Research, Haifa, Israel

Abstract. While Web Services ensure interoperability and extensibility for networked applications, they also complicate the deployment of highly collaborative systems, such as virtual reality environments and massively multiplayer online games. Quite simply, such systems often manifest a natural peer-to-peer structure. This conflicts with Web Services' imposition of a client-server communication model, vectoring all events through a data center and emerging as a performance bottleneck. We design and implement the Kevlar system to alleviate such choke points, using an overarching network-overlay structure to integrate central hosted content with peer-to-peer multicast. Kevlar leverages the given storage and communication models that best match the respective information: data most naturally retrieved from the cloud is managed using hosted objects, while edge updates are transmitted directly peer-to-peer using multicast. Here, we present the Kevlar architecture and a series of carefully controlled experiments to evaluate our implementation. We demonstrate Kevlar's successful and efficient support of deployments across wide-area networks and its adaptivity and resilience to firewalls, constrained network segments, and other peculiarities of local network policy.

Keywords: Distributed systems, Overlay networks, Collaboration

1 Introduction

The rapid evolution of the Internet has both produced and relied upon standards for interoperability, enabling client systems to easily interact with sophisticated data centers that host data, or create content. To leverage these standards, more and more collaborative applications — virtual reality immersion environments [15], massively multiplayer online games [9], conference systems [10], and other distributed tools — are now designed as Web Services [1], bouncing client-initiated events off central shared data centers for relay to other clients. While such an architecture functions acceptably in most settings, it performs and scales poorly in particular environments: when the link between some set of clients and the data center is congested, the latency on such a link is too high, or the servers in the data center are overloaded.

In prior work, we introduced a new method for combining standard Web Services with peer-to-peer replication protocols. The Live Distributed Objects (LO) platform [5,16] supports a simple drag-and-drop style of application development, similar to that used by non-programmers to design Web pages. The resulting applications have an XML representation suitable for sharing: for example, via e-mail or through a networked file system. Each user who "opens" an LO application activates the objects within some scope (for example, objects visible from some location in a game environment), and each of the resulting object instances function as a replica of what is conceptually a distributed object. We provide more details about our LO approach in Sect. 2, but, for now, note that the design of LO starts to address the issues listed above.

However, LO lacks the type of protocol capable of fully exploiting this flexibility, leaving open the question of whether our LO platform actually includes the full range of needed mechanisms. In particular, although LO is effective in enterprise network settings, the LO replication protocols are stymied by the many barriers that arise in practical Internet deployments across wide-area networks (WANs): performance variability, firewall interference, IP multicast (IPMC) policy non-uniformity, and more [14]. In the public Internet of 2010, point-to-point TCP is often the only option to work reliably (and, even that, at times only in one direction).

In this work, we introduce a substantially more capable wide-area multicast solution embodied by our Kevlar system. Kevlar allows applications to create multicast regions ("patches"), sewn together into a single spanning application-layer multicast structure, for content distribution or event-style notifications. Kevlar is a decentralized and improved re-implementation of our prior work, Quilt [12], restructured into a collection of Live Objects more robust to node failures. Here, we also apply Kevlar to demonstrate a substantial application: a search-and-rescue command-and-control system, useful in responding to environmental events such as hurricanes or earthquakes. We then undertake a careful evaluation of our solution, accurately emulating potential deployment scenarios and measuring associated event delivery latencies, data rates, and overhead.

Kevlar addresses a difficulty that arises from the need to construct applications that will run as a network of interconnected components and do so in a way sensitive to their local runtime environments. Here, we focus on the selection of a multicast component from among a set of multicast protocols, each specialized for a different setting, but the potential applicability is broader. While our research community has recognized that applications using component architectures should benefit from such improved structure, the issue of *adapting* such applications to match their runtime environments has received much less attention. Our work shows that this is a solvable problem, and, through our detailed evaluation, that the solutions are truly practical. The key enabler is the Kevlar platform architecture: it is structured to support plug-in components along with developer-provided rules, used at runtime to decide which component best matches a given runtime environment.

The primary contributions of this work are the following:

- WAN collaboration requires sophisticated management of firewalls, bottleneck links, organizational policy, and performance-driven protocol optimization. KEVLAR automates such tasks and shows that Web Services and peer-to-peer protocols can co-exist in Internet WAN environments, despite the many complexities that arise from network configurations and performance limitations. KEVLAR illustrates a new and interesting form of runtime adaption, in which a distributed application must determine which components to launch on the basis of runtime criteria evaluated on a per-node basis.
- KEVLAR applies the LO component model to replace the centralized features of our Quilt system with a completely decentralized, gossip-based solution, thus eliminating a central point of failure, as well as an additional performance bottleneck.
- We evaluate KEVLAR across a range of experiments within a challenging and complex network environment. In contrast to most popular Web-Service architectures, as well as our prior Quilt system, neither of which deliver acceptable or reliable performance in this scenario, KEVLAR succeeds in its design goals for efficient collaborative applications.

The KEVLAR system is distributed under a FreeBSD license, available for download and deployment from `http://kevlar.cs.cornell.edu/`. While the current version scales to hundreds of simultaneous users, our ongoing work expands support to configurations with tens of thousands or hundreds of thousands of users. To achieve such scalability, we are extending KEVLAR's multicast-forwarding components to allow for pre-emptive filtering of data during multicast. This capability would align KEVLAR more closely with the needs of distributed virtual worlds and online games and their high-volume, rich data streams: the server need only send all the data once, for each client to receive an optimal, personalized subset of the complete stream. Further discussion of such extensions is outside the scope of this work.

2 Background

In detailing the architecture of KEVLAR, we will first briefly summarize the Live Objects component model [5,16], upon which KEVLAR is built. LO presents a new kind of "live distributed object," replicated at each node in the system. The replicas can communicate among themselves with any choice of protocol, and the ensemble together represents the actual live object. These live objects can be composed as a graph, in which case, the replicas of each distinct live object on a given node interact using a simple, but flexible, event-based interface. Type checking is employed both at design time and at run-time, and the type-checking system itself is highly flexible. For example, an LO representation of an airplane might compose an airplane rendering object with an object that fault-tolerantly captures location data from a GPS source, which in turn is composed with a multicast protocol presented through an LO interface: a "data replication" object. The airplane might require protocols that support total ordering and real-time delivery, and the type checker would have to verify that the associated

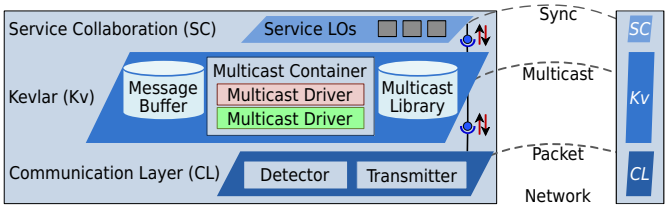


Fig. 1: ***Architecture of a Kevlar-enabled LO application.*** *Three major Live Objects compose a demonstration application: Service Collaboration (SC), Kevlar (KV), and Communication Layer (CL). As the application executes, all replicas of each LO communicate with one another across the network.*

protocols offer these properties (though the LO platform does not independently ensure such claims are correct).

As mentioned earlier, LO includes a drag-and-drop application builder. Many applications require little to no programming, simply using existing objects parameterized with URLs pointing to data sources. Additional objects simply implement a standard interface that we provide. This style of development would be familiar to any Java or C# user familiar with graphical interface design. The resulting LO application resides as an easily shared XML representation.

Figure 1 presents the architecture for a prototypical Kevlar application. Here, Service Collaboration (SC), Kevlar (KV), and Communication Layer (CL) live objects interface, on a given system node, through their respective event-based interfaces. Inside each live object (such as SC), we find a graph composed by several service objects (e.g., maps, airplanes, weather, etc.). In a distributed environment, the replicas for each service live object communicate with other replicas of their type across the network, sharing content or synchronizing state. Meanwhile, the KV and CL objects provide multicast function and real packet delivery on their own respective levels.

Prior work on the LO approach focused on type checking and event-driven interaction mechanisms [16] and efforts to scale through multicore parallelism [17]. But the actual LO multicast protocols have thus far been simple, rather limited, ones that operate only in enterprise data centers or other local-area networks among groups of computers without intermediate firewalls or performance barriers. As a result, prior to this work, all LO demonstration applications have been unsuitable for WAN deployment.

3 Patchwork Overlay in Kevlar

While Kevlar uses the LO component model, it also extends a more recent system of ours: Quilt distributed event-notification [12]. As a free-standing library, Quilt combines independent multicast patches for separate network regions into a single application-layer overlay across complex WAN settings. Figure 2 depicts this process for Kevlar, with a single overlay including WAN links and

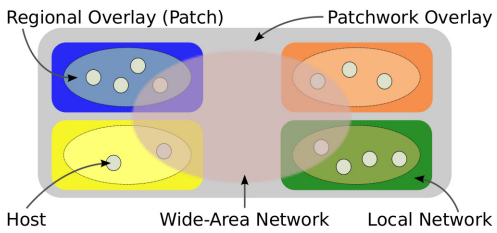


Fig. 2: ***Patchwork overlay.*** *Kevlar automates the construction of multicast overlays across complex environments, with hosts in various local and wide-area networks. Here, Kevlar combines a collection of regional multicast overlays (patches) into a single system-wide patchwork overlay, thus separating the multicast interface for the application from the runtime-driven, performance-optimized implementation of the overlay.*

Byte Offset	0	1	2	3	4	5	6	7	8	.....	23	24	25	26	27	28	29	30	31	...	34
EUID Content	Direction (d)	Protocol (p)	List of {d,p} tuples		# of routers		IPMC range		Router Stack (path of routers to local DNS)			Optional flag	Performance type (t)		Min value (l)		Max value (h)		List of {t,l,h} triples		
Type	Connectivity				Local Topology							Measured Performance									

Fig. 3: ***EUID.*** *Components of Kevlar's Environmental Unique IDentifier with accompanying byte offset (bar atop number signifies variable width of particular entry), enumerating metrics for connectivity, local topology, and measured performance.*

connecting four independent local networks, each with its own multicast policies, firewall placement, and congestion profile. As one might expect, there is no single, simple solution that can operate optimally in all such settings.

A KEVLAR application developer creates a set of multicast "drivers," each consisting of a protocol implemented as an LO, and an associated rule describing the conditions under which that protocol can be used. (The API for these driver rules is that of Quilt [12].) As earlier shown in Fig. 1, KEVLAR itself consists of a library into which these drivers are linked. At initialization of a KEVLAR application, the system profiles its local environment through a series of sophisticated measurements and updates the performance measurements continually during runtime. The resulting metrics are aggregated as the "environmental unique identifier" (EUID) in Fig. 3, which is used for assignment of nodes to a particular patch.

During application initialization, KEVLAR nodes send their EUIDs to a bootstrap server and obtain a list of other nodes that appear to be "close" to the node with respect to the topology, latency, bandwidth, and protocol compatibility, as indicated by the EUID. Unlike Quilt, which relies on a central server to incrementally construct and maintain the patchwork, KEVLAR nodes create and maintain patches in a peer-to-peer fashion. Quilt's single point of failure is thus eliminated, since the bootstrap functionality is trivial. This decentralized

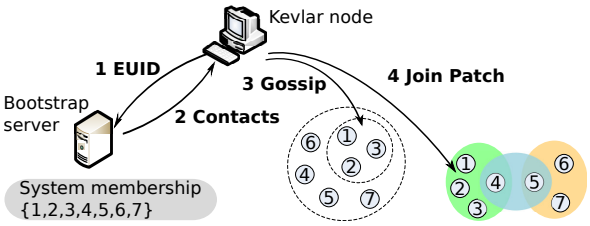


Fig. 4: ***Patch assignment.*** *Upon initialization, a Kevlar node tests its local environment to produce an EUID; (1) sends this EUID to a bootstrap server; (2) receives a contact list in return; (3) uses this list to identify other nearby members and patches via gossip; (4) applies these EUID metrics to dynamically select patches to join.*

scheme also improves security and privacy, as domains have the option of running protocols and networking rules privately with no dependence on an external Quilt server.

In a further departure from Quilt, KEVLAR runs a system-wide anti-entropy gossip protocol [18] to exchange information about membership, EUIDs, and patches that are formed, along with their identifiers and the protocols in use. For our experiments, we set the gossip rate to one exchange per second. The choice of a gossip recipient is biased towards proximate nodes, according to the EUID information. More specifically, we compose a fixed-length portion of the EUID into a number that reflects protocol compatibility, presence on the routing path (shorter routing paths are padded with zeros), average latency, and bandwidth. We then define the approximate *distance* between two nodes to be the absolute difference between these numbers mapped from their EUIDs; thus the above order of composition reveals the priority of each component in determining proximity and compatibility between end-hosts. In each gossip round, we rank the recipients by the distance and select the recipient of rank i with probability proportional to 2^{-i}. This results in membership sets biased toward nearby nodes; hence, updates in the node's surrounding environment are disseminated faster.

KEVLAR seeks to maximize connectivity while relaying multicasts between patches in a fault-tolerant manner. It also avoids duplicate event delivery, even under node churn. Figure 4 illustrates patch formation and assignment logic (details as in Quilt [12]).

The KEVLAR multicast protocols used in this paper include:

1. **IP multicast** (IPMC) [7]. This protocol assigns a distinct IPMC class-D multicast address to each region. The associated rule checks to make sure that IP multicast is permitted by examining all network interfaces used by each application, and testing their IPMC-enabled bit.
2. **DONet** [21]. This protocol implements a mesh-structured application layer multicast that uses TCP for the links. The actual content is disseminated on a random-graph overlay maintained by a gossip protocol. It works in the style of BitTorrent [6]: every node advertises its local content buffer

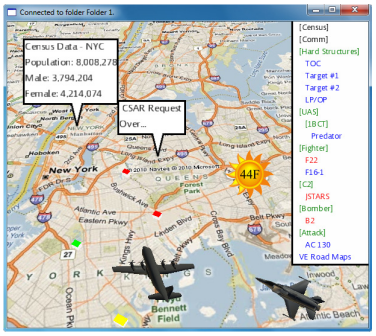


Fig. 5: ***Example application.*** *This Kevlar demonstration, showcasing a search-and-rescue application, requires significant communication traffic among cooperating users at the network edge as well as constant query and retrieval of centralized data objects.*

snapshot epidemically with a fixed fanout; then, by the end of each scheduled epoch, it solicits non-received data in a bandwidth-aware manner; once a request is received, the node returns the associated data, also in a bandwidth-aware manner. KEVLAR uses DONet for physically close end-users without IPMC support, since such users have lower latencies between them, varied bandwidth capacities, and more churn than data center servers.

3. **OMNI Tree** [4]. The OMNI tree is a latency-optimized application layer multicast (ALM) overlay, serving both multicast receivers and Multicast Service Nodes (MSN), which are actually local proxies. Assuming MSN nodes have negligible latency, 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. KEVLAR uses OMNI Tree as the inter-patch protocol to sew different patches together. For fast and stable forwarding among regional patches, the associated rule prefers nodes with accessible connections and high network performance.

4 Anatomy of a Kevlar Application

As a canonical demonstration of KEVLAR, we implement a distributed search-and-rescue application which exercises almost all of the functionality described above. Figure 5 depicts a representative user interface, displaying both centrally hosted and edge objects and allowing for user manipulation of the scenario. Such an application could readily be designed and assembled by a non-programmer, using pre-existing Live Objects (stored, for example, in a local or shared file system) that each represent a different piece of data: aircraft, ground vehicles, rescue workers, weather information, maps, etc. The developer simply synthesizes the scenario from pertinent objects and associates them with their data sources. Further changes to the application can be made at runtime, by any of the users sharing the system (provided, of course, that they have appropriate

permissions). For instance, commanders may modify the positions of aircraft or adjust the demographic statistics displayed on the interface. Components can also be reused, even simultaneously: those involved in Fig. 5 could separately participate in any other sort of KEVLAR application.

The user accesses this application through its XML representation, which includes references to the "live folders" that hold content added at runtime. Thus, the user interface can be viewed as a visualization of the theoretical "graph" of program composition, expressed in its XML representation. While the overall graph structure is determined by the original application designer, some portions are, by design, extensible by other users at runtime: the live folders just mentioned, for example. The "leaves" of the graph are the airplanes, maps, weather, etc. Some leaf objects extract and display content from cloud repositories, while others use peer-to-peer protocols (for example, to track locations of moving objects). As the graph evolves (either through movement of objects, or their explicit addition or deletion), the interface will automatically update in sync.

Much of the functionality behind this user interface requires communication, using multicast channels established by KEVLAR. These channels retrieve hosted content, share edge updates, or coordinate or synchronize some other action. For example, satellite imagery might be updated via multicast to all users as a satellite over-flies a region of interest. A first responder in the field could, similarly, use a multicast group to share a status report on some disaster victims; his unit commander could multicast orders to a larger rescue unit, and so forth.

The KEVLAR runtime is implicitly launched through the execution of associated applications. As determined by the KEVLAR application developer, a given application includes some set of built-in multicast protocols (for our experiments here, the three defined in Sect. 3). For each node and multicast group, KEVLAR determines the appropriate protocols to launch and helps these protocols form initial peering relationships with other members of a given node's patch. Beyond this, each protocol maintains its own peer structure. Thus, where IPMC is permitted and other peers are also present, KEVLAR will launch the IPMC protocol modules and provide them with class-D multicast addresses as parameters, as well as a few contacts for initialization. Using an OMNI network, KEVLAR will sew this IPMC region together with other remote regions: KEVLAR will launch the OMNI protocol and help the module peer with appropriate remote nodes. On nodes that run both OMNI and IPMC protocols, KEVLAR will help relay multicasts between them, minimizing the risk of network partition through redundant dual-protocol nodes, while simultaneously suppressing the duplicate delivery of updates. If failures occur or nodes quit the overlay, KEVLAR orchestrates the repair of the mesh, all in a fully decentralized manner.

5 Evaluation

5.1 Experiment Setup

The remainder of this paper presents an experimental evaluation of our KEVLAR system in a number of distinct application scenarios.

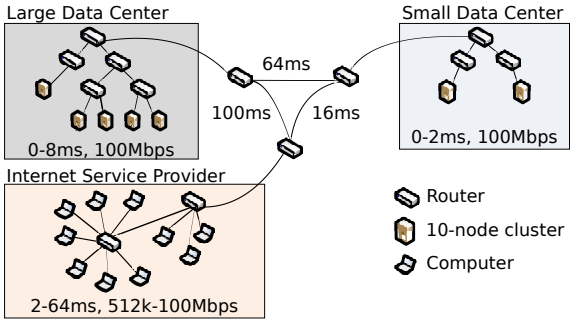


Fig. 6: ***Network topology.*** *Experimental network topology, as emulated on DETERlab, with data centers and an Internet service provider: node numbers, latencies, and data rates as indicated, with IP multicast internally enabled in data centers.*

We first explore the ability of KEVLAR, compared to other multicast protocols, to establish an efficient overlay topology and minimize latencies from a single given sender to the ensemble of receivers. Further, we investigate the bandwidth utilization in such a topology, comparing KEVLAR to the native OMNI Tree and DONet protocols, as well as to a pure IP multicast (which, of course, might not actually be a legal choice in settings where KEVLAR is launched, but represents something of an ideal when permitted). We explore a number of various simple application scenarios, each characterized by a set of input traffic parameters of fixed message size and data rate; these tests reveal the applications for which KEVLAR's quality of service determination yields improved results. We continue our systematic exploration of KEVLAR's performance for complex real-time collaborative applications by considering the needs of the search-and-rescue application discussed earlier, and illustrated in Fig. 5. Finally, we conclude our evaluation by examining the robustness of KEVLAR to recover from catastrophic failure cases.

We rely upon DETERlab [8], an Emulab [20] environment, to construct our experimental environment and establish a network topology that includes various types of local subnets. In total, we use 80 nodes, distributed among three different regions and connected across an emulated Internet backbone. As shown in Fig. 6, the regions consist of: (1) a large data center whose internal LAN has three layers of depth to its hierarchy; (2) a smaller data center with a 2-layer LAN; and (3) a consumer ISP, remote from big data center but topologically near the small one; the ISP supports consumers via various last-mile access technologies (including cable, DSL, and satellite, each with appropriate latency, jitter, and bandwidth constraints). IP multicast is enabled inside data centers, but unusable for home users. Bandwidth and latency settings accompany Fig. 6. The object of this topology is to accurately mimic a portion of the real Internet, including some large data centers hosting the primary server farm and working in concert with small data centers distributed around the world to provide low-latency support for nearby users.

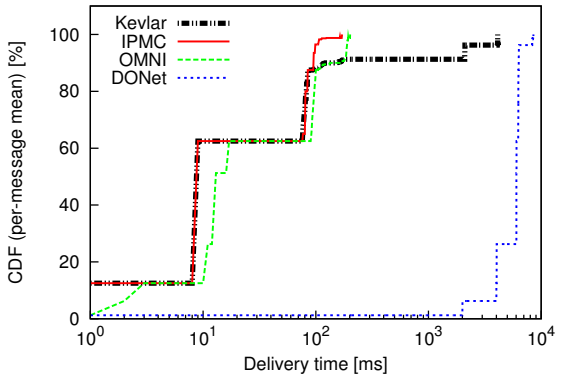


Fig. 7: ***Delivery latency.*** *Cumulative distribution function (CDF) of the latency for a given data-center sender to contact some percentage of all other nodes (in experimental topology in Fig. 6), for each of four competing protocols: mean computed over ten messages; Kevlar closely tracks IPMC except for the final 15% of the nodes (which, as part of the ISP, cannot access IPMC), where it instead uses the bandwidth-efficient, but higher-latency, DONet protocol.*

Unless otherwise specified, all data points correspond to an average over ten trials. Error bars are calculated as one sample standard deviation but omitted from figures when too small to be clearly visible.

5.2 Examination of Overlay Topology

We first examine the raw performance of Kevlar along with the various alternative protocols in the overlay topology without the influence of particular traffic patterns of any specific application. We accomplish this measurement by sending a series of small messages (10-Byte payloads) at low data rates (100 messages per second), so that the available bandwidth across the links and data transfer delays do not affect these results.

Finding low-latency paths. First, we evaluate the efficiency of each protocol in matching its overlay to the actual physical network topology. Any efficient multicast service should identify fast delivery pathways to each receiver irrespective of the application traffic input. Figure 7 shows just such a comparison among IPMC, DONet, OMNI Tree, and Kevlar — the latency for a sender to send data to all the receivers in the experiment environment. Here, and in the ensuing sections, we use the cumulative distribution function (CDF) to display many of our measurements; the structure of many of the dissemination patterns incorporate bursts of traffic, such that the associated probability density function (PDF) would be filled with spikes that are difficult to interpret visually.

A number of qualitative observations emerge from Fig. 7. First, we note that our Kevlar multicast protocol is closest to what IPMC offers (recall that IPMC is generally unavailable in a real WAN deployment, hence it represents an ideal, but one that may not be an actual option). Both show a largely stair-stepped

Table 1: ***Forwarding load.*** *Comparison of the skew in load among four competing protocols, demonstrating both the average forwarding load per system node, as a percentage of the traffic stream, as well as the number of nodes that forward traffic, among all eighty system nodes. With network hardware support, IPMC requires only a single forwarding node, while OMNI Tree and DONet must rely on many intermediate nodes. Kevlar nodes generally only forward one-fifth of the packets they receive.*

Protocol	Load per node [%]	Forwarding nodes [#]
IPMC	1.3	1
KEVLAR	20.3	12
OMNI Tree	100.0	20
DONet	102.8	66

CDF that corresponds to (1) low-latency communication in the large data center from which the sending traffic originates; (2) fast transfer to the smaller data center; and (3) final delivery to small, remote ISP. We also note the differentiation between KEVLAR and IPMC for delivery to the final 15% of nodes, located in the remote ISP; this is due to the lack of IPMC among the ISP customers (which therefore receive *no* IPMC traffic), and KEVLAR's reliance there on the underlying DONet protocol (which utilizes bandwidth better than OMNI Tree for such end-host users). To understand the performance of the OMNI Tree protocol, we must recall that it cannot, by design, leverage IPMC even within the data centers where it is available and must instead build its own tree structure to disseminate data. This consumes time and thus accounts for CDF values that are universally lower than (or equal to) that of IPMC (and KEVLAR, again by design). Finally, we see that DONet is the least capable of the protocols in creating an efficient overlay. As explained above in Sect. 3, DONet's protocol uses only "pull" semantics (rather than "push" in the other protocols) and this introduces significant latency to identify the node location that possesses the required data. DONet's use of epochs and timers only exacerbates this problem (though, in general, as we see later, the use of epochs has some benefits in enabling better bandwidth utilization).

Forwarding load on overlay nodes. Recall that some intermediate nodes in an ALM may need to bear the burden of forwarding packets to other receivers. We explore the efficiency of overlay formation for the multicast protocols by examining the average volume of traffic that each node in the overlay needs to transmit to distribute a given stream, as a fraction of the total traffic volume.

Table 1 shows this amount of traffic that must be forwarded by an average overlay node, as well as the number of nodes that forward packets. With IPMC, only a single sender needs to forward packets, so, on average, nodes transmit only a fraction (1/80) of the stream. As expected, the OMNI Tree and DONet protocols do not benefit from network-level multicast and thus the typical node

must forward every packet received. KEVLAR is second to IPMC in performance: an average node forwards 20% of packets; this indicates that KEVLAR is able to exploit IPMC where supported and to use an ALM (DONet) otherwise.

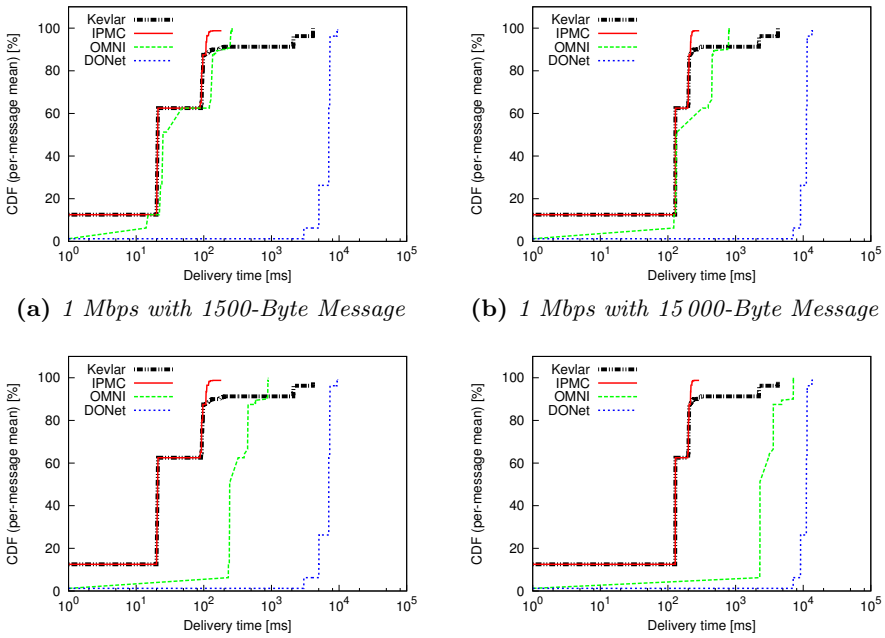
5.3 Performance of Simple Applications

This section focuses upon the realized delivery performance among a number of applications with simple, fixed traffic patterns. Specifically, we measure the performance with respect to fifteen different ensembles of traffic streams, each with different fixed values for message size (150, 1500, and 15 000 Bytes) and data rate (100 kbps, 300 kbps, 1 Mbps, 3 Mbps, and 10 Mbps). As mentioned before, these streams represent five different levels of media production: standard-quality audio MP3, network video conferencing, Video CD (VCD), standard-definition (SD) IP Television (IPTV), and high-definition (HD) IPTV [11]. In each scenario, we contrast the four protocols in terms of their CDF, determined by message delivery times averaged across ten measurements.

Figure 8 shows our results, here. We make three primary qualitative observations from this series of measurements: The first point is that all of these subfigures show the same qualitative structure for each protocol, considered independently, as we found in the application-agnostic evaluation from Fig. 7. We would expect as much — the qualitative features of the performance, for a particular application (with a given fixed data rate and message size) using a specific multicast protocol, are largely determined by the corresponding overlay. However, for each particular application, the curves for the individual protocols show various offsets in time as well as dilation or contraction of features in time. Next, for delivery to a large fraction ($\gtrsim 90\%$) of the receivers, KEVLAR shows identical (ideal) performance when compared to IPMC within all application scenarios. For dissemination to the final fraction of nodes (those within the ISP), KEVLAR out-performs DONet in all cases, while it exceeds OMNI Tree only for 15 000-Byte messages at 10 Mbps (see Fig. 8(d)). Lastly, we observe no qualitative distinctions among the measurements conducted at data rates of 100 kbps, 300 kbps (for either message size) and those of 1 Mbps (additional tests at 3 Mbps and for 150 Byte messages gave identical results). Thus, due to space constraints, we do not reproduce the former in Fig. 8.

We now separately examine the behavior of each protocol within the same measurements conducted above, and previously shown in Fig. 8. This allows us to discuss more subtle distinctions in protocol performance at varying data rate and message size parameterization, that are otherwise not apparent above. Figure 9 presents a comparison of the IPMC, OMNI Tree, DONet, and KEVLAR protocols for different application domains (different fixed input parameters).

Two quantitative observations emerge: In contrasting data rates of 1 and 10 Mbps (for all message sizes), we note that the curves representing IPMC, KEVLAR, and DONet protocols are completely unaffected by data rate considerations. (In fact, such curves identically overlap each other, to the extent that we remove such extra labels within each subfigure to increase legibility.) Conversely, the OMNI Tree protocol collapses with increasing data rates: the curves for the



(a) *1 Mbps with 1500-Byte Message*

(b) *1 Mbps with 15 000-Byte Message*

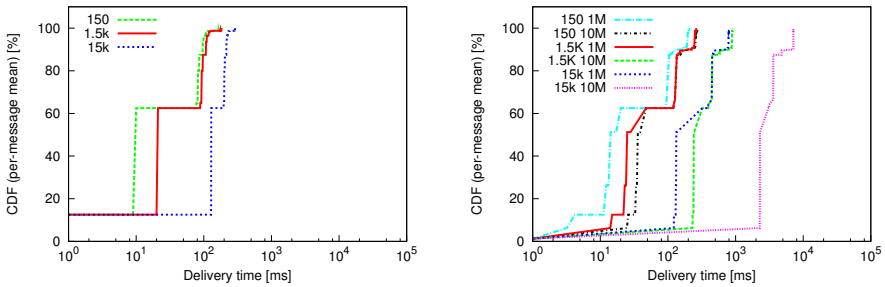
(c) *10 Mbps with 1500-Byte Message*

(d) *10 Mbps with 15 000-Byte Message*

Fig. 8: ***Performance of simple applications (input traffic dependence).*** *Performance comparison of multicast protocols for input traffic of various message size and data rate: each subfigure, corresponding to a different set of input parameter values, shows the CDF (per-message mean as in Fig. 7) of message delivery time. Space constraints preclude visualization of the remaining eleven experiments (specifically, data rates of 100 kbps, 300 kbps and 3 Mbps, and message size of 150 Bytes), with nearly identical results.*

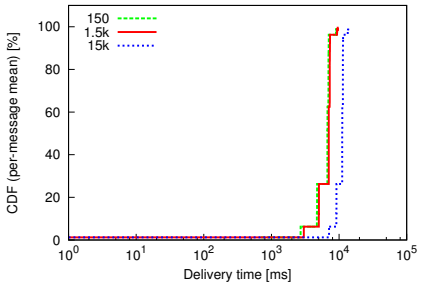
higher rates are shifted to the right in time by factor of ratio between rates (10 in figure). The impact of data rate on OMNI Tree is suitably severe that its performance for smaller messages at higher data rates (1500 Bytes at 10 Mbps) is worse (or equal, at times) than that for larger messages at lower data rates (15 000 Bytes at 1 Mbps). We conjecture that this effect is due to the inefficiency of bandwidth utilization by the OMNI Tree protocol.

We also contrast message sizes of 150, 1500, and 15 000 Bytes for all data rates and note a key qualitative difference: The curves for IPMC, KEVLAR, and OMNI Tree protocols are all shifted to the right by an order of magnitude in time, while DONet is significantly less affected by this variation in message size parameter. This observation follows trivially due to the need to transfer larger amounts of data to the receivers. The smaller correlation between message size and DONet performance can be explained by DONet's parallelization of data

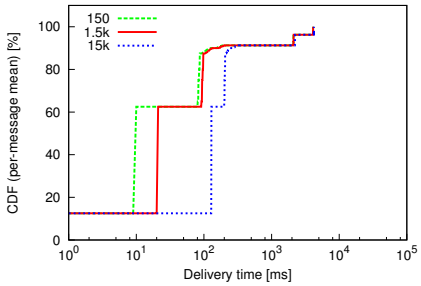


(a) *IPMC protocol*

(b) *OMNI Tree protocol*



(c) *DONet protocol*



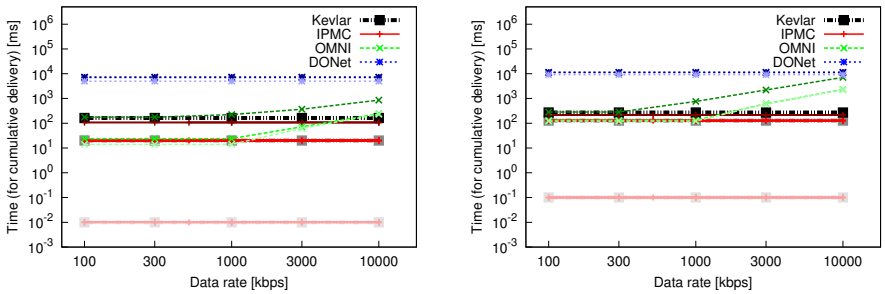
(d) KEVLAR *protocol*

Fig. 9: ***Performance of simple applications (protocol dependence).*** *Performance comparison of multicast protocols for input traffic of various message size and data rate: each subfigure, corresponding to a different protocol, shows the CDF of delivery time. (Note, for protocols with data-rate independent results, only message size is shown.)*

transfers for small portions of each message amongst many nodes (within each epoch that it defines).

Finally, Fig. 10 extracts the performance trends from the raw data presented in Figs. 8 and 9. Now, for each of the four multicast protocols studied, we consider the time required to distribute application traffic to various cumulative subsets of the entire environment. Thus, the family of curves for each protocol is comprised of three separate curves, which represent the delivery to different cumulative values (10%, 50%, and 90%) of the total node population; increasing color saturation of each curve in the family denotes more complete dissemination of traffic.

This figure further confirms many of the trends we discussed above: (1) strong dependence on input traffic data rate for OMNI Tree; (2) correlation between input message size and delivery time for IPMC, KEVLAR, and OMNI Tree (weak for DONet); and (3) for IPMC and KEVLAR protocols, the faster relative increase (more vertical CDF) in distribution for 15 000-Bytes messages as compared to 1500-Byte messages; this is seen in the narrower spacing within a given family of curves in Fig. 10(a) as compared to Fig. 10(b).



(a) *1500-Byte message size* **(b)** *15 000-Byte message size*

Fig. 10: ***Overall trends for simple application delivery.*** *Performance of multicast protocols for transmission of various traffic streams (each with fixed message size and data rate) associated with different video delivery standards. Subfigures show distinct message sizes (1500 and 15 000 Bytes, respectively), and, as a function of data rate, each plots the time required to distribute messages to cumulative subsets of the entire environment: the three separate curves, in each protocol family, display the time corresponding to different cumulative values (10%, 50%, and 90% delivery in the corresponding CDFs of Fig. 8). Space constraints preclude display of third subfigure with five additional curves, associated with 150-Byte packets at all five possible data rates.*

5.4 Performance of a Complex Application

Above, we have just evaluated the performance for a few simple, model applications that exhibit constant traffic patterns with fixed message size and data rate. Now, we expand upon such an evaluation by discussing the performance implications from diverse traffic patterns of a type associated with more complicated applications, focusing on the search and rescue application discussed in Sect. 2.

In Fig. 11, the inset depicts the input traffic pattern for this scenario. It plots message size as a function of time over the 10 minute period of application execution and communication. (We note that the data rate, here, is a secondary feature based upon the density of messages in time; however, we present the data in this manner, rather than as a 2D histogram, so as to reveal the time correlations of data, which would be absent in the histogram presentation.) This input figure depicts various traffic patterns generated by users' operations: during the start up, the collaborative application requires a checkpoint of existing service objects, generating a burst of messages followed by low traffic during local resource initialization; after a short time, users are able to explore the world map and add or remove new services of their choosing, all resulting in bulk transfers of texture contents from the data center; after fetching all the needed data, communications among users are steady and moderate, primarily at the network edge.

The main panel of Fig. 11 shows the resulting performance for each protocol in this scenario. We first compute CDFs associated with all four protocols,

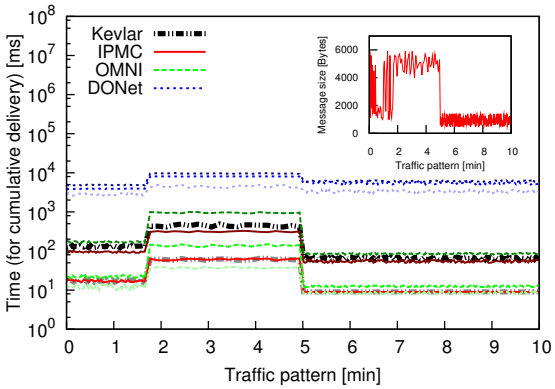


Fig. 11: ***Performance of a complex collaborative application.*** *Performance for multicast protocols while executing a complex collaborative application: inset shows the varying input traffic pattern of the application, with message size as a function of the position in time in the pattern. For each protocol, the main panel displays a family of curves plotting the time required to distribute messages to cumulative subsets of the entire environment, as a function of position in the input traffic pattern (increasing color saturation corresponds to cumulative delivery as in Fig. 10).*

for each of the (dozens of) individual sample points — representing a distinct value of data rate and message size — as seen in the inset of Fig. 11. From each of these CDFs, we then extract the time for the protocols to achieve different cumulative distributions of 10%, 50%, and 90%. Figure 11 thus shows four families of curves, one for each of the protocols, with each family differentiating the behavior associated with various completion levels in the dissemination of data. Here, increasing saturation levels of each curve color denote more completion of traffic distribution.

We make four observations, here: (1) KEVLAR tracks IPMC very closely, for all completion levels up to, but not including 90%. This reflects KEVLAR's underlying usage of IPMC within both data centers, its need for DONet in the non-IPMC-enabled ISP, and its use of a small-sized OMNI Tree for sewing its multicast patches together. (2) Throughout this experiment, across all varying message sizes and data rates, OMNI Tree shows a consistent performance penalty of approximately 400%. (3) DONet is the slowest of the protocols here, requiring seconds for the traffic to arrive at a significant fraction of the receivers. Indeed, its performance is degraded 100-fold, compared to both IPMC and KEVLAR; as discussed above, this primarily results from the its inclusion of a handshake scheme, an epoch-based time line, and a "pull"-based communication pattern. However, in fairness to DONet, we recognize that its performance variance upon varying data rate is significantly lower that that of OMNI Tree; this meets its design goal to optimize for bandwidth utilization instead of latency. (4) Finally, we examine the difference in time for each protocol to achieve 50% versus 90% cumulative distribution of traffic: First, we note that DONet shows little differ-

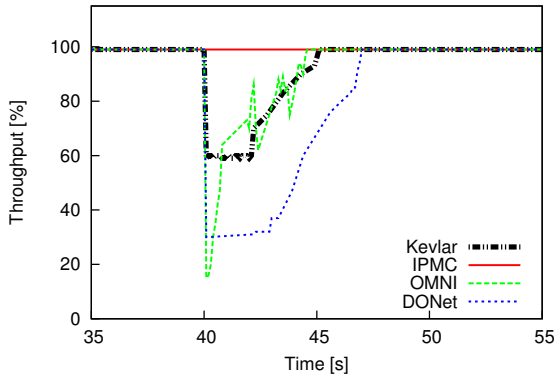


Fig. 12: ***Robustness under failure.*** *Robustness of the various multicast protocols upon fail-stop (at 40 seconds) of a random 50% of the nodes during transmission of 1500-Byte messages at 1 Mbps data rate: figure plots percentage throughput to remaining live nodes relative to the throughput prior to failures, as a function of time.*

ence between these curves; we expect as much from previous measurements (see Fig. 8) in which the DONet curves show very abrupt transitions from negligible completion to almost total distribution of traffic. Conversely, IPMC, OMNI Tree, and KEVLAR all show an approximate 5-fold increase in time required to reach 90% of receivers compared to that needed to deliver to 50% of receivers. Lastly, we note that KEVLAR's relative performance with respect to the ideal of IPMC is not as strong for delivery to 90% of nodes as it is for 50%; this is due to KEVLAR's reliance upon non-IPMC underlying protocols for its ISP transmission as compared to its dissemination within each data center.

5.5 Robustness

We conclude our evaluation by considering the robustness of KEVLAR in contrast to other multicast protocols. To obtain this measurement, we introduce catastrophic failures in a large portion of the nodes within our environment and monitor the resulting recovery for each multicast protocol. Specifically, we select the scenario from Sect. 5.3 with 1500-Byte messages sent at a data rate of 1 Mbps; after steady, non-perturbed communication for 40 seconds, we fail-stop a random selection of 50% of the nodes. (To ensure that each protocol confronts the identical failure scenario, we perform the random selection of nodes only once, and then reuse the same choice of failed nodes for the measurements of all the different protocols.) In quantifying the robustness of each system, we compute the percentage throughput received by the remaining live nodes relative to the amount before onset of node failures.

Figure 12 depicts the robustness results for this measurement. We immediately observe that IPMC performs ideally, showing no loss of throughput. All other protocols suffer a dramatic reduction in throughput as their overlays are

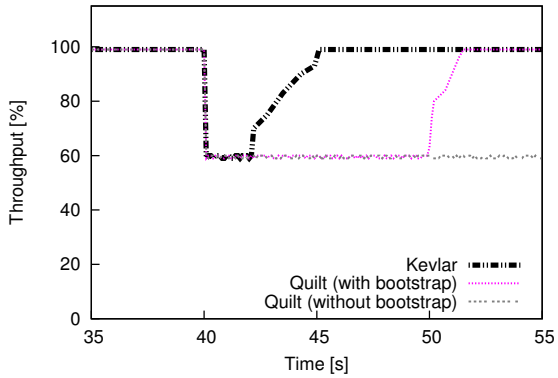


Fig. 13: ***Kevlar's improvement in robustness over Quilt.*** *Comparison of robustness of Kevlar's distributed patchwork maintenance against Quilt patchwork implementations, both with and without centralized bootstrap servers (same scenario as in Fig. 12). Kevlar shows a 100% improvement in recovery time versus Quilt with bootstrap servers; without bootstrapping, Quilt never recovers from the failure scenario.*

disturbed by node failures; this is to be expected for ALMs, in contrast to physical IPMC. We observe that the OMNI Tree protocol has the largest loss of throughput; this likely results from the failure of high-level (root) nodes in its tree structure that then disrupt delivery to all the nodes in their sub-hierarchy. Recovery of OMNI Tree is relatively quick, though; its event-driven rejoin procedure reconstructs the tree overlay faster than DONet's epoch-style protocol recovers its overlay. We also observe that OMNI Tree shows fluctuations in its recovery as it is attempting to simultaneously optimize its structure for latency considerations while new nodes are also rejoining the overlay. The DONet ALM is slower to recover than OMNI Tree; however, since traffic load is balanced across its nodes, DONet throughput does not ever decrease as dramatically as that of OMNI Tree. Finally, we consider Kevlar and observe that it outperforms both OMNI Tree and DONet in terms of robustness: Kevlar maintains higher throughput than either other ALM, as it can still utilize the underlying IPMC protocol in patches were it is available. Further, Kevlar recovers as quickly as the OMNI Tree protocol. Finally, during recovery, Kevlar's throughput monotonically increases, unlike the fluctuations during OMNI Tree's overlay reconstruction.

We perform one additional measurement to quantify Kevlar's enhanced robustness. In Fig. 13, we contrast Kevlar against two different versions of our Quilt [12] protocol, as introduced in Sect. 1 above. Here, we include Quilt both with and without its use of a bootstrap server. In the centralized Quilt framework, we observe that the bootstrap system assures recovery, at a cost of increased recovery time due to greater load on a few central bootstrap servers. Kevlar's use of distributed patch maintenance shows at least a 100% improvement in recovery time compared with Quilt.

5.6 Summary of Measurement Evaluation

We now summarize our measurements and observations from this section: we showed that KEVLAR empirically achieves a performance envelope matching our design goals articulated earlier. The KEVLAR system consistently leveraged whichever protocol was most efficient among those available in each topological network patch. Obviously, if IPMC were available in all situations, simply using IPMC might be the best plan. But when IPMC is not available, KEVLAR is a very practical and effective alternative.

6 Related Work

KEVLAR's object-oriented design was inspired by Quality Objects (QuO) [3,19]. The QuO middleware provides quality-of-service information to allow programs to function reasonably well on WANs and LANs. During run-time, QuO applications may alter modes, between "safe," "overloaded," or "graceful shutdown," causing objects to change their behavior accordingly.

We are unaware of technologies, other than KEVLAR and Quilt that use environmentally aware patchwork overlays for multicast. Patchwork overlays for other purposes, however, have been proposed: A large-scale routing system, MONET [2], groups together different kinds of client links into "patches" to allow IP packets to traverse NATs and firewalls and to optimize inefficient application-level routing paths. Similarly, OCALA [13] accommodates legacy applications over modern network architectures by combining different overlays to reach disadvantaged network hosts.

7 Conclusions

KEVLAR delivers a flexible architecture for creating collaborative applications that can be efficiently utilized in the wide-area Internet. The system encourages modular extension and facilitates development of sophisticated applications through composition of distributed objects to form graphs, within which object instances interact by event-passing.

Typical users of collaborative applications, such as virtual reality platforms and massively multiplayer online games, are subject to quite non-uniform Internet environments. Thus, runtime adaptation is required to appropriately configure these deployed applications. Indeed, KEVLAR automates selection and activation of the correct component out of a set of functionally similar options, each optimized for different conditions.

KEVLAR innovates at several levels: through its flexible and modular architecture; through the mechanisms used to select appropriate components; and through the optimization-driven decision layers that create the desired distributed infrastructure. This structure controls what might otherwise be a very complex system. A careful evaluation shows that KEVLAR really does function as designed.

Acknowledgments We are grateful to the Chinese National Science Foundation (Project No. 607-31160730), National Science Foundation, Air Force Research Laboratory, EU IST Project CoMiFin (FP7-ICT-225407/2008), Intel Corporation, and Cisco Systems for their support of this research.

References

1. Gustavo Alonso, Fabio Casati, Harumi Kuno, and Vijay Machiraju, *Web Services Concepts, Architectures and Applications*, Springer, 2004.
2. David G. Andersen, Hari Balakrishnan, M. Frans Kaashoek, and Rohit N. Rao, *Improving Web Availability for Clients with MONET*, Proc. of NSDI '05 (Boston, MA, USA), 2005.
3. Michael Atighetchi, Partha P. Pal, Christopher C. Jones, Paul Rubel, Richard E. Schantz, Joseph P. Loyall, and John A. Zinky, *Building Auto-Adaptive Distributed Applications: The QuO-APOD Experience*, Proc. of ICDCSW '03 (Washington, DC, USA), 2003.
4. S. Banerjee, C. Kommareddy, K. Kar, B. Bhattacharjee, and S. Khuller, *Construction of an efficient overlay multicast infrastructure for real-time applications*, Proc. of INFOCOM 2003 (San Francisco, CA, USA), 2003.
5. Ken Birman, Jared Cantwell, Daniel Freedman, Qi Huang, Petko Nikolov, and Krzysztof Ostrowski, *Edge Mashups for Service-Oriented Collaboration*, IEEE Computer **42** (2009), no. 5, 90–94.
6. Bram Cohen, *Incentives Build Robustness in BitTorrent*, Tech. report, 2003.
7. Stephen E. Deering and David R. Cheriton, *Multicast routing in datagram internetworks and extended LANs*, ACM Trans. Comput. Syst. **8** (1990), no. 2, 85–110.
8. DETERlab, `http://www.isi.deterlab.net/`.
9. Wu-chang Feng, David Brandt, and Debanjan Saha, *A Long-Term Study of a Popular MMORPG*, Proc. of NetGames '07 (Melbourne, Australia), 2007.
10. Google Voice and Video Chat, `http://www.google.com/chat/video/`.
11. HDV Specification, `http://www.avchd-info.org/format/`.
12. Qi Huang, Ymir Vigfusson, Ken Birman, and Haoyuan Li, *Quilt: A Patchwork of Multicast Regions*, Proc. of DEBS '10 (Cambridge, UK), 2010.
13. Dilip Joseph, Jayanth Kannan, Ayumu Kubota, Karthik Lakshminarayanan, Ion Stoica, and Klaus Wehrle, *OCALA: An Architecture for Supporting Legacy Applications over Overlays*, Proc. of NSDI '06 (San Jose, CA, USA), 2006.
14. Tom Leighton, *Improving Performance on the Internet*, Commun. ACM **52** (2009), no. 2, 44–51.
15. Frederic P. Miller, Agnes F. Vandome, and John McBrewster, *Second Life*, Alpha Press, 2009.
16. Krzysztof Ostrowski, Ken Birman, Danny Dolev, and Jong Hoon Ahnn, *Programming with Live Distributed Objects*, Proc. of ECOOP '08 (Paphos, Cypress), 2008.
17. Krzysztof Ostrowski, Chuck Sakoda, and Ken Birman, *Self-Replicating Objects for Multicore Platforms*, Proc. of ECOOP '10 (Maribor, Slovenia), 2010.
18. Robbert van Renesse, Yaron Minsky, and Mark Hayden, *A Gossip-Based Failure Detection Service*, Proc. of Middleware '98 (The Lake District, UK), 1998.
19. Rodrigo Vanegas, John A. Zinky, Joseph P. Loyall, David Karr, Richard E. Schantz, and David E. Bakken, *QuO's runtime support for quality of service in distributed objects*, Proc. of Middleware '98 (The Lake District, UK), 1998.
20. Brian White, Jay Lepreau, Leigh Stoller, Robert Ricci, Shashi Guruprasad, Mac Newbold, Mike Hibler, Chad Barb, and Abhijeet Joglekar, *An Integrated Experimental Environment for Distributed Systems and Networks*, Proc. of OSDI '02 (Boston, MA, USA), 2002.
21. Xinyan Zhang, Jiangchuan Liu, Bo Li, and Tak-Shing Peter Yum, *CoolStreaming/DONet: a data-driven overlay network for peer-to-peer live media streaming*, Proc. of INFOCOM 2005 (Miami, FL, USA), 2005.