An Open Content Delivery Infrastructure Using Data Lockers

Richard Alimi[◇] Lijiang Chen[†] Dirk Kutscher[‡] Hongqiang Harry Liu[◇]
Akbar Rahman[+] Haibin Song[⊙] Yang Richard Yang[◇] David Zhang[×] Ning Zong[⊙]
[◇]Yale [⊙]Huawei [†]HP Labs China [+]InterDigital [×]PPLive [‡]NEC Laboratories Europe

ABSTRACT

Content distribution has become a major function of the Internet. However, the current Internet content distribution infrastructure is largely closed to end-to-end applications, making it challenging for the application community to utilize in-network storage resources. In this paper, we investigate a simple paradigm named SAILOR that introduces application-definable, shared in-network *data lockers* to effectively facilitate the construction of highly efficient, cooperative content distribution applications. We design and implement a prototype of SAILOR and integrate it with two popular content distribution applications for file and live streaming respectively. Our experimental results clearly demonstrate that SAILOR can significantly improve both network efficiency and application performance, thereby benefiting both network providers and application providers.

Categories and Subject Descriptors: C.2.1 [**Computer Communication Networks**]: Network Architecture and Design–Network communications.

General Terms: Performance, Reliability.

Keywords: Content Delivery, Open In-Network Storage.

1. INTRODUCTION

Motivation. As content distribution becomes a major function of the Internet [4], the scalability, efficiency, and flexibility of Internet content distribution applications can have significant impacts on the operation of the Internet.

A minimum approach of constructing content distribution applications is to use mostly the resources of participating (sending/receiving) end hosts, but this approach can have inherent limitations. Consider an application session with N end users receiving an object of size S. Without in-network resources or support, a purely end-host based approach requires that the endhosts provide at least $N * S$ upload traffic over the last-mile access networks. However, access networks can be the major performance and cost bottlenecks for many network service providers.

Both Internet service providers (ISP) and content delivery network (CDN) providers have deployed in-network resources to assist content distribution, but the existing designs are closed to applications. Specifically, ISPs may deploy transparent [19] and/or non-transparent forward caches [18] at strategic locations of their networks; CDNs deploy edge servers at data centers and/or inside some ISP networks. However, for either ISP or CDN deployed in-network resources, in the current design, applications have limited capability to control the usage of in-network resources (*e.g.*, replication sites, the time of replication, the resource allocation among end users). As a result, many applications with a large content delivery component choose to deploy their own infrastructures. Such infrastructures can be costly and represent a major barrier for application deployments.

Our approach. In this paper, we design a paradigm named SAILOR (simple, application-independent data lockers) to provide a shared in-network storage infrastructure for effective integration with large-scale, end-to-end content distribution applications. In particular, SAILOR provides end users and/or content publishers in-network *data locker* accounts. A specific content distribution application, with the resource account owner's authorization, uses application-specific information and state to directly control and make the best use of the available resources at the in-network data lockers. In-network data lockers have the potential to significantly lower the cost of entrance to content distribution and thus generate innovative solutions by the large application development community.

We show that integrating SAILOR with content distribution applications can provide flexibility and efficiency in addressing a wide range of issues facing both networks and network applications in content distribution.

Specifically, SAILOR can significantly improve efficiency of network resource usage (*e.g.*, inter-domain, backbone, and access network usages). For example, by uploading from data lockers instead of clients behind access networks, as in the minimum approach, SAILOR substantially reduces access network resource usage. In our experiments, we show that the deployment of SAILOR by one cable ISP can reduce its access network upload by about 44%, and up to 96% if SAILOR clients are extensively deployed at other ISPs. SAILOR/Live reduces access network median upload rate by up to 95%. By integrating SAILOR with intelligent peering, we also demonstrate substantial reduction of ISP backbone and inter-domain traffic, thereby further benefiting ISPs. For example, SAILOR/File reduces inter-domain supply fraction by up to 87% and backbone bandwidth-distance product by 58%; SAILOR/Live reduces backbone traffic supply fraction by up to 95%.

Our experiments also clearly demonstrate the benefits of SAILOR to application performance. In particular, SAILOR/File accelerates the file download rate by 23%, shifting application bottleneck rate from access network uplink bandwidth to downlink bandwidth. SAILOR/Live reduces the startup delay of live streaming to 1/3 at the 80-th percentile, thereby improving user experience.

Paper organization. The remainder of the paper is organized as follows. In Section 2, we give an overview of the SAILOR architecture entities. In Section 3, we present the guiding design requirements of SAILOR. In Section 4, we discuss key data locker design features. We present the integration of SAILOR with two content delivery applications in Section 5. In Section 6, we evaluate the performance of SAILOR. We survey related work in Section 7, and conclude in Section 8.

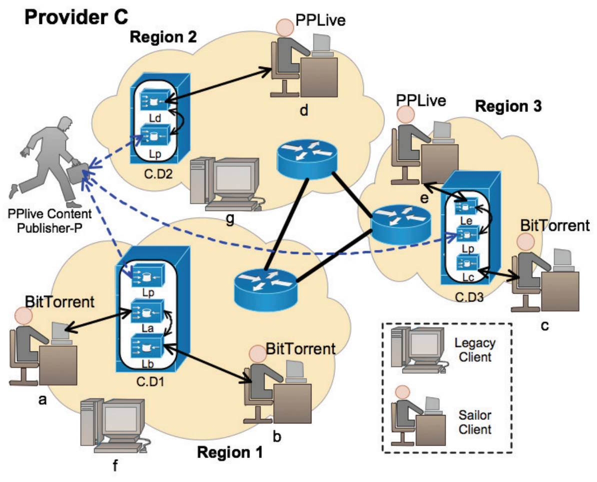


Figure 1: An example: provider C provides multiple data locker servers to end users and content publishers.

2. ARCHITECTURE ENTITIES

SAILOR is a simple architecture that introduces shared in-network content delivery data storage infrastructures into the design of applications. In the SAILOR architecture, there are multiple data locker service providers. Each data locker service provider has one or multiple data locker servers, which are hosted at different network locations such as cable modem termination system (CMTS), wireless base stations, and/or data centers of different regions. Figure 1 shows a data locker service provider C that has deployed multiple data locker servers.

Each data locker server partitions its resources to create multiple data lockers. Each data locker is assigned to one user, and requires the user's permission in order for other users to access. A user can be either an end user or a content publisher. We use L_a to denote the data locker of a user a. Each data locker L_a has an access key K_{L_a} for implementing access control and resource usage constraints.

Each data locker service provider makes its own business and policy decisions about who can get an account or accounts and which of its data locker servers to use. For example, it can grant accounts to consumer subscribers (*e.g.*, along with Internet service), or it may have a policy to grant accounts only to trusted content publishers to reduce the problem of potential illegal content.

Usage scenario I. As an example deployment scenario, consider an ISP C that provides data lockers to its subscribers. Figure 1 shows that C operates three data locker servers at three locations. The data locker of a subscriber should be hosted on a close-by data locker server that belongs to C. For example, L_a is hosted at data locker server D_1, which is close to subscriber a.

Subscriber a can use L_a in many ways. For example, a can use L_a for file sharing. Specifically, subscriber a can first download a piece, say *File.fileid.piece1*, to L_a. Then, while downloading the piece to its local machine, a can start to simultaneously upload the piece to its peers b and c from L_a instead of from its local machine. By uploading from L_a, this approach saves access network bandwidth and thus can avoid congestion on the bottleneck uplink. By overlapping the downloading to a and the uploading to b and c, this approach speeds up content distribution control cycle.

Concurrent with the file-sharing session, subscriber a may be also watching a live video or participating in a multi-party video conference. In either case, a can also use L_a to reduce the usage of bottleneck resources.

Usage scenario II. Figure 1 shows that C also can allocate to content publisher p one data locker on each of the three data locker servers. Content publishers can benefit from multiple data lockers by performing application-specific replication and achieving increased proximity to users. Specifically, the content publisher p, say a streaming application provider, can proactively distribute the blocks of popular channels to its data lockers. Then the content publisher guides users to download from its data lockers to substantially improve distribution efficiency, without having to construct its own dedicated infrastructure.

3. DESIGN REQUIREMENTS

Despite the preceding specification on architecture entities, there is still a large design space on data lockers. To guide our design, we first present the key requirements.

Simple end-to-end control. The most important design requirement of SAILOR is the consistent application of the end-to-end design principle [21]. Users and their applications can have more information on the need and impacts of resource allocation. Hence, users and applications provide guidance on resource allocation decisions and the infrastructure implements the decisions. Specifically, when an application client is downloading data from others, it decides whether to download to the client local host directly, to its locker only, or first to its locker and then to the local host. Similarly, when an application client is uploading data to others, it decides who can receive its data, and whether to upload from the locker or directly from the end host. Also, the application client decides how much resources (*e.g.*, the number of connections, network bandwidth) that a data request consumes.

A corollary benefit of the end-to-end control is decoupled control and data functions. Control functions tend to be processing intensive, whereas data functions are often storage/bandwidth intensive. The decoupling between control and data functions not only allows us to optimize control and data functions separately, yielding better overall efficiency, but also permits more flexible, open and evolvable control platforms on top of a shared data infrastructure.

Efficient and secure infrastructure storage. One design possibility is complete virtualization, by considering each data locker as an isolated container of the user's content data. However, such an isolation-based design is inefficient in infrastructure resource usage and does not take advantage of a shared content distribution infrastructure provided by SAILOR. In particular, by considering each data locker as an isolated data container, there can be substantial amount of data redundancy. In the worst case, when N users are interested in a common object, the object has to be fetched and stored N times. On the other hand, blindly sharing all data is undesirable either, because it may be important for some content distribution applications to protect the ownership and the privacy of the data. Therefore, it is essential for SAILOR to provide efficient and secure data sharing.

Multiplexing gain for infrastructure resources. A major potential benefit of SAILOR is significant inter-user statistical multiplexing gain in the consumption of critical in-network resources such as bandwidth. By aggregating demands from a large number of users and provisioning them collectively, SAILOR should require significantly less resources than provisioning for each individual user separately, so long as the individual users' resource demands do not peak at exactly the same time.

In addition, SAILOR should allow intra-user multiplexing gain. A typical storage system has a provider-defined quota for each user, and the user is responsible for allocating resources within the quota. However, this is not sufficient for content distribution applications. A user may run multiple applications sharing resources in the data locker. Since it may not be practical to allow independent applications (*e.g.*, a live-streaming application and a file-sharing application, from two different providers) to jointly allocate resources (*e.g.*, by introducing a protocol between them), the resource model must allow applications to manage resources independently. A natural solution is to allow the user to define a simple per-application quota. However, this may waste available resources in a user's data locker.

Low-latency, scalable data locker control. One major difference between a traditional application and a content distribution application is that the content distribution application often needs a large number of connections, requires frequent resource adjustment (*e.g.*, connect or disconnect peers, reallocate bandwidth share among peers), and must be sensitive to control overhead due to the real-time nature of content distribution. On the other hand, with an in-network storage infrastructure like SAILOR, resources are consumed at a remote location and cannot be directly observed. Also, data locker servers may become a bottleneck to resource management. Thus, it is desirable to reduce the number of control messages sent between an application and its data locker.

4. DATA LOCKERS DESIGN

The preceding design requirements impose challenges but at the same time provide substantial guidance to our design. Specifically, to design a shared network storage infrastructure for integration with large-scale end-to-end applications for efficient content distribution, we need to specify the following three components: (i) the *resource model*; (ii) the *data model*; and (iii) the *data locker access*. Below, we introduce the resource model and data locker access protocol of SAILOR.

Resource model: SAILOR uses a simple, general, weight-based resource model to partition each resource, unless it is overridden by another more specific policy. The model applies to all three types of resources, including the number of open network connections, network bandwidth, and storage space. The simple, weight-based resource model provides a simple, flexible mechanism to achieve the multiplexing-gain requirement.

Specifically, let $w_a(R)$ be the weight assigned to entity a for resource R. Then a's share of resource R is

$$\max\left(U_a(R), R_{total}\frac{w_a(R)}{\sum_{b \text{ is active}} w_b(R)}\right),$$

where the first term ($U_a(R)$) is a guaranteed amount of resource R to entity a; the second term is the amount of resource R computed for entity a, considering the relative weights of all active entities. If not many entities are active, an active entity a can get more than the guaranteed $U_a(R)$, as SAILOR uses work-conserving scheduling. An application may compute the value of $U_a(R)$ to guarantee basic performance (*e.g.*, base rate streaming), and then take advantage of additional amounts of resources made available by weighted sharing (*e.g.*, by using dynamic streaming). Note that providing resource guarantee requires an admission control module.

Figure 2 shows a 3-level hierarchy used in SAILOR. At the highest level of the hierarchy, a resource R is partitioned among concurrent users according to their subscription levels at the data locker service provider. This is consistent with multiple recent proposals on applying user-based fairness [10]. At the middle level, a user assigns each application a weight. This scheme solves the issue of distributed application resource coordination. That is, a user may have multiple independent applications running simultaneously on the user's local machine or across multiple machines (*e.g.*, multiple machines at a user's home), and these applications should share the

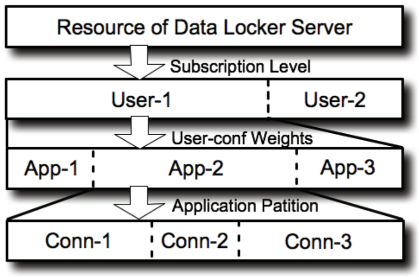


Figure 2: Hierarchy of Resource Allocation.

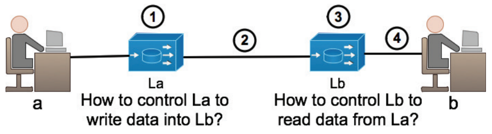

1
2
3
4
a
La
How to control La to write data into Lb?
Lb
How to control Lb to read data from La?
b

Figure 3: A scenario illustrating the need for distribute.

resources of a single user. Note that the scheme does not require any third party middleware to handle resource sharing among applications. At the lowest level, an application partitions the resource among the connections, if applicable and necessary. Controlling bandwidth share at a per-connection level is important for fairness schemes such as the proportional-share incentive scheme [15, 25].

Data locker access protocol: SAILOR provides a low-latency, light-weight implementation of the resource model. Specifically, applications issue commands to data lockers to store and retrieve data. The basic command in SAILOR is the distribute primitive, which enables efficient, pipelined multi-segment data access. Given the wide deployment of HTTP for content distribution, the encoding of distribute is based on HTTP.

One might think that there is no need to use a new primitive; instead we can treat all of the data lockers collectively as a distributed file system (similar to NFS) and simply apply the traditional file system access commands: write to store objects, and read to retrieve objects. However, such an approach is insufficient to support latency-sensitive, bandwidth-intensive content distribution applications. Consider a scenario shown in Figure 3. In this scenario, user a has previously stored an object in its locker L_a, and decides to send the object to a peer b. According to the end-to-end principle, peer b may decide to first download the object to its locker, L_b, and then download from its locker. The distributed file system view of data lockers can have major difficulties in this scenario.

A distribute request from a client C to a data locker server S specifies (i) a data object identifier dataid, (ii) a source node src, (iii) an account on S S:account, and (iv) a destination node dst. So a request distribute(dataid, src, S:account, dst) signals the distribution of a data object dataid from src to S:account to dst. Note that the src and dst fields can be empty. If src is empty, it becomes a pure read request to transfer data from S:account to dst; if dst is empty, it is a pure write request to store data from src to S:account. Figure 4 illustrates a use case of distribute.

The distribute primitive is designed to explicitly support pipelined multi-segment data access. Specifically, each data object in SAILOR consists of a sequence of *segments*. As soon as the data locker server S receives a segment of an object from src, S can immediately send the segment to dst without waiting for the entire object to be received. Two or more distribute requests can be easily cascaded to enable multi-hop pipelined data access, to mimic a multi-hop CDN functionality, with intermediate reflectors.

The distribute primitive also supports distribution deduplica-

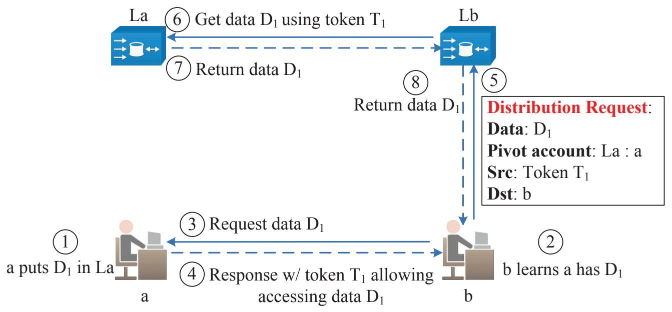


Figure 4: The distribute primitive.

```
distribute(dataid, src, S:account, dst)
1. if (no data across server for dataid) then
2.     server S gets data from src
3. elseif (src is an account on S)   // local dedup
4.     link data, and ack success
5. else                               // remote BW dedup
6.     challenge src to verify owning dataid
```

Figure 5: Algorithm for distribution deduplication.

tion. Specifically, to reduce duplications in content distribution, a server S can conduct distribution deduplication. Figure 5 depicts the deduplication algorithm.

Note that distribution deduplication allows us to efficiently implement point-to-multipoint distribution (*i.e.*, multicast) using multiple point-to-point distribute requests. Specifically, in order to distribute content from src to S:account to dst_k ($k = 1, \cdots, N$), we just need to issue N point-to-point distribute requests: distribute(dataid, src, S:account, dst_k) ($k = 1, \cdots, N$). The data locker server S can automatically eliminate redundant data transmission and storage.

The distribute primitve includes not only access tokens but also resource tokens: access tokens indicate permission to read/write data, and resource allocation tokens indicate resources allocated to perform the operation. Specifically, a resource allocation token includes the following:

$$(SIZE_{\max}, BW_{\text{weight}}, BW_{\max}, \text{ConnSlots}, \text{AccessTokenHash}).$$

The first three fields ($SIZE_{\max}$, BW_{weight} and $BW_{\max}$) of a resource token indicate the maximum number of bytes allowed to be transferred, a weight, and the maximum bandwidth allocated, respectively. The next field (ConnSlots) of a resource token indicates *the connection slots*.

The last field (AccessTokenHash) of a resource token optionally links the resource allocation token to a particular access token. An application may typically pass both an access token and linked resource allocation tokens to a peer resource allocation parameters. However, in some application scenarios, such as flash crowds, a client may have data that it wishes to upload to a peer, but it may lack sufficient resources to do so. This is one possible reason for poor performance during flash crowds.

5. APPLICATION/SAILOR INTEGRATION

Application controlled data lockers provide substantial flexibility and efficiency. Nevertheless, achieving the full benefits of SAILOR depends on effective integration of SAILOR with specific applications. In this section, we discuss two application-integration issues.

Backbone and interdomain traffic reduction. An objective of introducing data lockers is to significantly improve network efficiency. By reducing backbone and interdomain traffic, we reduce global traffic, and improve network efficiency.

Recently, multiple peering techniques (*e.g.*, Ono [3], and P4P [26]) have been proposed to reduce backbone and interdomain traffic. Although different in details, these techniques are based on using network costs to guide peering. One may also approximate such costs using third-party databases such as the Maxmind database [16].

Adopting these techniques to data lockers involves a single cost transformation. Consider the peer selection algorithm of a peer (leecher) a who has a data locker L_a. Let B be the set of potential peers that a can connect to. Given network costs without considering data lockers, we can derive network costs considering the effects of data lockers. As an example, let C^0_{ab} be the network cost when peer a sends to $b \in B$ without data lockers; let C_{ab} be the cost considering the effects of data lockers. We can derive C_{ab} based on C^0_{ab} and the effects of data lockers. There are three cases:

- If b is a legacy client, we have $C_{ab} \leftarrow C^0_{ab}$.
- If b is data locker capable but has no data locker, we have $C_{ab} \leftarrow C^0_{L_a b}$. If an ISP assigns a higher cost to upload from access networks, $C^0_{L_a b}$ can be much lower than C^0_{ab}.
- If b has a data locker, we have $C_{ab} \leftarrow C^0_{L_a L_b}$.

We can similarly derive the cost of C_{ba}.

Congestion control and reliability. Although applications are encouraged to use data lockers when available, there are settings when the data lockers are down, do not have enough capacities (*e.g.*, congested), and/or are on congested network paths. Thus, a key integration issue is how an application utilizes both data locker resources and end host resources. This is important for reliability and particularly important for content distribution applications with deadline constraints (*e.g.*, live streaming or video-on-demand).

An application can use a priority scheme to try to use data locker resources first, but can also adapt to use end host resources as backup. Consider the setting in Figure 3. An ideal setting is that b does a distribute request with source being L_a and destination b. But there can be four congestion/failure points (see figure for the points). If the ideal setting times out, user b tries to utilize resources in the following order: L_b, L_a, and a. Figure 6 gives the details of the priority algorithm.

```
priority-req(objid, Lb, a, La)
1.if (Lb && La && Lb is idle) then
2.    indicate Lb to download objid from La
3.elseif (La && La is idle) then
4.    request objid from La directly
5.elseif (objid is urgent)
6.    // the object is critical for app performance
7.    request objid from a directly
8.else
9.    hold the request until next schedule time
```

Figure 6: Algorithm to handle congestion/failure.

6. EVALUATIONS

We have conducted preliminary evaluation on the effectiveness of SAILOR through real experiments.

6.1 SAILOR FileSharing Results

We first integrate SAILOR with the popular BitTorrent file-sharing application.

Setting. We use a real trace collected at a commercial BitTorrent variant. The trace contains sufficient fields to allow us to assign clients to areas of networks of two major US ISPs, named ISP-A and ISP-B. We assume that only ISP-B provides data lockers to its users. We consider three deployment scenarios: Native, All SAILOR, and ISP SAILOR. In Native, no client is SAILOR capable (understands SAILOR protocol); in All SAILOR, all clients are SAILOR capable, although only clients inside ISP-B have data lockers; in ISP SAILOR, only clients inside ISP-B are SAILOR capable and have data lockers.

Network performance. Figure 7(a) shows the results for access supply. We observe that in All SAILOR, access upload supply is reduced by 96% compared with Native. In ISP SAILOR, the reduction is smaller because non-SAILOR peers are not able to retrieve data stored in lockers. Still we observe a reduction of 44% compared with Native. Comparing SAILOR with current techniques, such as Ono/P4P, we observe that they cannot reduce any access supply, while SAILOR/FileSharing significantly reduces it.

Figures 7(b) and (c) evaluate reduction on interdomain and backbone usage, respectively. ISP-B prefers intradomain peers to interdomain peers, and also assigns costs such that nearby intradomain peers are preferred. Integrating network costs in peer selection allows SAILOR to utilize more network-efficient peers. We observe that SAILOR/FileSharing clients reduce the interdomain supply from 83% to 11%. SAILOR also reduces the bandwith-delay-product (BDP) from 3.36 to 1.39 (58% reduction).

Application performance. Next, we evaluate the effects of SAILOR on application performance. Figure 8 shows the results. We observe that SAILOR/FileSharing improves median download rate from 320 kbps to 520 kbps, a 62% increase. Note that in this evaluation, SAILOR refers to the case of ISP SAILOR.

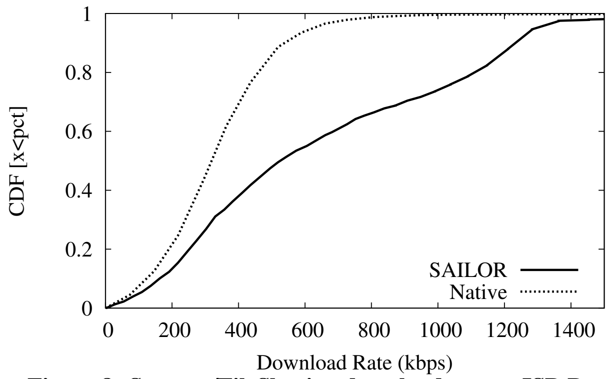


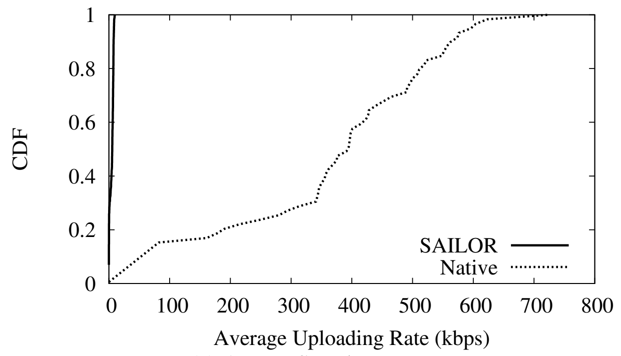
Figure 8: SAILOR/FileSharing download rate at ISP-B.

6.2 SAILOR Live Results

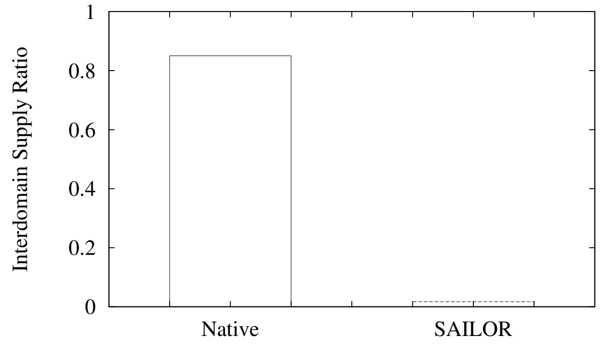
The preceding experiments are for file sharing. We now turn to live streaming to demonstrate SAILOR's effectiveness across application types.

Setting. We run real experiments on Planetlab. We utilize 400 planetlab nodes running PPLive, a production live streaming application. We run two experiments: native PPLive (Native) and PPLive with SAILOR (SAILOR/Live or SAILOR). The channel rate is set to 384 kbps in both experiments. For the SAILOR/Live experiment, we use a four-domain setting of Amazon EC2 with one data locker server at each domain; each client is assigned a data locker according to its location. For a fair comparison, the capacity of a data locker server is equal to the sum of the upload capacities of the clients assigned to it during the Native experiment.

Network performance. We start by showing that SAILOR significantly reduces access supply. From Figure 9(a)), we observe that the media access supply rate of SAILOR/Live is only 5.5 kbps while Native is 395 kbps. This result is not surprising since control messages are sent through access uplink. Heavy volume data traffic comes from in-network data lockers.



(a) Access Supply Rate



(b) Interdomain Supply

Figure 9: SAILOR/Live network efficiency.

SAILOR/Live also significantly reduces interdomain usage. Figure 9(b) shows the reduction of both interdomain supply ratio. Specifically, SAILOR/Live reduces interdomain supply ratio from 85.1% to around 1.67%, achieving as much as 98% reduction compared with Native. Distributed deduplication, in particular remote bandwidth deduplication, plays a major role in the reduction.

Performance Metrics	Improvement with SAILOR
Startup Delay	At 80-percentile: reduced to $\frac{1}{3}$ of Native
Piece Lost Rate	About the same, at $\leq 0.02\%$

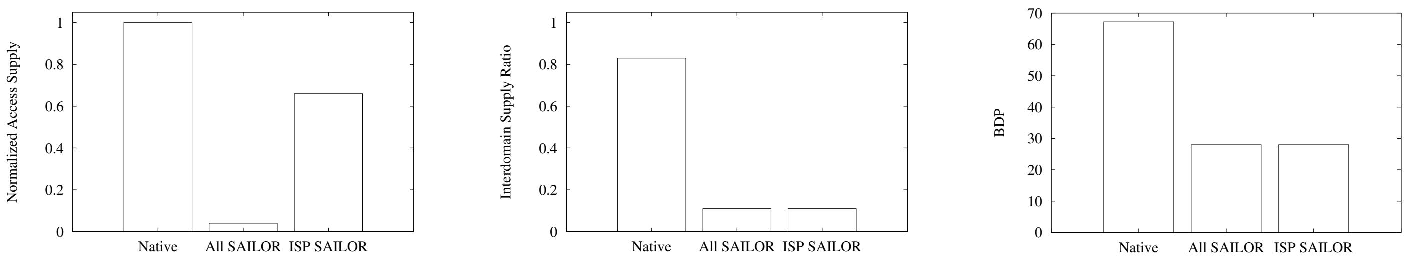
Table 1: Improvement to application performance by SAILOR (Planetlab).

Application performance. SAILOR reduces network traffic and at the same time improves application performance. Performance improvements using SAILOR are summarized in Table 1. For startup delay, SAILOR/Live reduces it to only $\frac{1}{3}$ of the Native at the 80-th percentile. The piece miss rates of the two experiments are similar, both at around 0.02%. A key reason for the speedup of startup delay by SAILOR is the multiplexing gain, as we discussed in the BitTorrent experiment. Furthermore, a SAILOR/Live peer starts to announce piece availability only after the piece has arrived at its locker, instead of waiting until it is at the last mile. This reduces content bottlenecks during flashcrowd.

7. RELATED WORK

Our work is directly motivated by the significant technical challenges in prior efforts to provide infrastructure support for end-to-end content distribution applications (*e.g.*, [7–9, 12, 14, 17, 20, 22–24]).

One way to implement SAILOR is to adopt a generic network storage service such as Oceanstore, FreeNAS, or Amazon S3. In [1, 2], Beck *et al.* propose Internet Backplane Protocol (IBP), a middleware created to allow the sharing of storage resources. Although encouraging, none of these aforementioned network storage systems are designed for end-to-end content distribution applications.



(a) Access Supply (b) Interdomain Supply (c) BDP

Figure 7: SAILOR/FileSharing efficiency for ISP-B.

There is no previous study on how to design a network storage infrastructure for integration with end-to-end, large-scale content distribution applications.

There are substantial interests in adapting the Internet architecture for content-oriented architectures that focus more on name-based routing; see [5, 11, 13, 27] for some sample projects of the recent literature, which is too large to review here. SAILOR focuses on resource sharing and decoupling of control and data forwarding in content delievry networks.

Our project draws on many discussions in the IETF DECADE working group [6]. Instead of focusing on architecture, we focus on the resource model, the data access primitive, and evaluations.

8. CONCLUSIONS AND FUTURE WORK

We present SAILOR to provide a simple, shared, application-independent in-network storage infrastructure for effective integration with large-scale, end-to-end content distribution applications. SAILOR introduces key features that are essential for designing effective content distribution applications using network storage. Through effective integration with two major applications, we demonstrate that SAILOR can provide substantial benefits to both network service providers and applications.

For our future work, we are planning large scale trials with major ISPs and application vendors. We are also evaluating potential extensions including access lockers of inactive users and shared lockers. As end users' computing and data move more into the networks, data locker service can become an integral part of the Internet infrastructure.

9. ACKNOWLEDGEMENTS

H. Harry Liu and Leo Chen are supported in part by a gift from Huawei. We are grateful to James Chen, Xiaohui Chen, Yixuan Geng, Yingjie Gu, Lili Qiu, Ye Wang, Hanyu Wei, Rich Woundy, Harvey Xue, Peng Zhang, Yin Zhang, and Yunfei Zhang for collaborations on the project. Many participants from the IETF DECADE Working Group made contributions to this project.

10. REFERENCES

[1] A. Bassi, M. Beck, T. Moore, J. S. Plank, M. Swany, R. Wolski, and G. Fagg. The Internet Backplane Protocol: A study in resource sharing. *Future Generation Computing Systems*, 2003.

[2] M. Beck, H. Liu, T. Moore, and Y. Zheng. Pipelining and caching the Internet Backplane Protocol. Technical Report UT-CS-04-529, University of Tennessee, 2004.

[3] D. R. Choffnes and F. E. Bustamante. Taming the torrent: A practical approach to reducing cross-ISP traffic in P2P systems. In *Proc. of SIGCOMM*, 2008.

[4] Cisco Visual Networking Index (VNI). Hyperconnectivity and the approaching zettabyte era, June 2010. `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/VNI_Hyperconnectivity_WP.html`.

[5] C. Dannewitz. NetInf: An information-centric design for the future Internet. In *Proc. 3rd GI/ITG KuVS Workshop on The Future Internet*, 2009.

[6] IETF DECADE Working Group. `http://www.ietf.org/html.charters/decade-charter.html`.

[7] K. Gummadi, R. Dunn, S. Saroiu, S. Gribble, H. Levy, and J. Zahorjan. Measurement, modeling, and analysis of a peer-to-peer file-sharing workload. In *Proc. of SOSP*, 2003.

[8] M. Hefeeda, C. Hsu, and K. Mokhtarian. pCache: A proxy cache for peer-to-peer traffic. ACM SIGCOMM'08 Technical Demonstration.

[9] M. Hefeeda and B. Noorizadeh. Cooperative caching: The case for P2P traffic. In *Proc. of LCN*, 2008.

[10] IETF P2Pi Workshop. `http://www.ietf.org/internet-drafts/draft-p2pi-cooper-workshop-report-00.txt`, 2008.

[11] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *Proc. of CoNEXT*, 2009.

[12] T. Karagiannis, P. Rodriguez, and K. Papagiannaki. Should Internet service providers fear peer-assisted content distribution? In *Proc. of IMC*, 2005.

[13] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A data-oriented (and beyond) network architecture. In *Proc. of ACM SIGCOMM*, 2007.

[14] N. Leibowitz, A. Bergman, R. Ben-Shaul, and A. Shavit. Are file swapping networks cacheable? characterizing P2P traffic. In *Proc. of WCW*, Boulder, CO, 2002.

[15] D. Levin, K. LaCurts, N. Spring, and B. Bhattacharjee. BitTorrent is an auction. In *Proc. of SIGCOMM*, 2008.

[16] MaxMind. `http://www.maxmind.com/`.

[17] A. Nakao, K. Sasaki, and S. Yamamoto. A remedy for network operators against increasing P2P traffic: Enabling packet cache for P2P applications. *IEICE Transactions on Communications*, 2008.

[18] Oversi. `http://www.oversi.com/index.php?option=com_content&task=view&id=48&Itemid=103`.

[19] PeerApp. `http://www.peerapp.com/products-ultraband-How-Caching-Works.aspx`.

[20] O. Saleh and M. Hefeeda. Modeling and caching of peer-to-peer traffic. Technical Report TR 2006-11, SMU, 2006.

[21] J. Saltzer, D. Reed, and D. Clark. End-to-end arguments in system design. *ACM Transactions on Computer Systems*, 1984.

[22] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy. An analysis of Internet content delivery systems. In *Proc. of OSDI*, Boston, MA, Dec. 2002.

[23] G. Shen, Y. Wang, Y. Xiong, B. Y. Zhao, and Z.-L. Zhang. HPTP: Relieving the tension between ISPs and P2P. In *Proc. of IPTPS*, Bellevue, WA, Feb. 2007.

[24] A. Wierzbicki, N. Leibowitz, M. Ripeanu, and R. Wozniak. Cache replacement policies revisited. In *Proc. of GP2P*, Chicago, IL, 2004.

[25] F. Wu and L. Zhang. Proportional response dynamics leads to market equilibrium. In *Proc. of STOC*, 2008.

[26] H. Xie, Y. R. Yang, A. Krishnamurthy, Y. Liu, and A. Silberschatz. P4P: Provider portal for applications. In *Proc. of SIGCOMM*, 2008.

[27] L. Zhang and V. Jacobson. Named data networking (NDN) project, Oct. 2010.