On the Design of a Practical Information-Centric Transport

Victor Grishchenko, Flutra Osmani, Raul Jimenez,
Johan Pouwelse, Henk Sips
j.a.pouwelse@its.tudelft.nl

Completed March 2011.

Report number PDS-2011-006

ISSN 1387-2109

Published and produced by:
Parallel and Distributed Systems Group
Department of Software and Computer Technology
Faculty Electrical Engineering, Mathematics, and Computer Science
Delft University of Technology
Mekelweg 4
2628 CD Delft
The Netherlands

Information about Parallel and Distributed Systems Report Series:
`reports@pds.ewi.tudelft.nl`

Information about Parallel and Distributed Systems Section:
`http://www.pds.ewi.tudelft.nl/`

Abstract

A recent chain of exploratory publications proposed information-centric architectures for the Internet. The common pitfall of such proposals is the imbalance of upfront costs and immediate benefits. To address this concern, we focus on prospects of piecemeal adoption. We start with the *necessary* basic primitives of any infocentric architecture, primarily self-certifying names, to determine what are they *sufficient* for. We define natural interfaces to other parts of the architecture, primarily naming and routing, for which no special assumptions are made. We define a natural separation of infocentric transport and internetworking layers and their message vocabulary, that allows to run our infocentric transport over IP, UDP, TCP, HTTP or entirely IP-free. As a proof of concept, we have implemented a UDP-based transport protocol named *swift* with per-datagram data integrity checks. Our architecture highly prioritizes modularity and sketches a path for piecemeal adoption which we consider a critical enabler of any progress in the field.

Victor Grishchenko, Johan Pouwelse and Henk Sips are employed by Delft University of Technology. Flutra Osmani and Raul Jimenez are employed by the Royal Institute of Technology, Isafjordsgatan 39, Kista, Sweden, {`flutrao, rauljc`}`@kth.se`.

Contents

List of Figures

List of Tables

1 Introduction

During the past years we saw a chain of exploratory publications considering the information-centric (also “named-data”, “content-centric” [27], “data-oriented” [25]) networking paradigm. The historical conversation-centric end-to-end model, embodied in the TCP/IP stack, is based on message exchange between pairs of network nodes, typically “servers” and “clients”. In the named-data architecture, the nodes directly address the data they need, independently of its source or its current place of storage. In a sense, named-data network breaks with the end-to-end abstraction, as there are no “ends” and the entire network is considered a “cloud”, which both stores and serves the data.

The idea of named data is not entirely new. To a large degree, it is in line with the *de facto* state of the things in modern networks. Over 90% of today’s Internet bandwidth [6] is effectively devoted to disseminating static multimedia content. There are some past, current, and developing technologies which might be described as “covertly” information-centric. For example, the historical Usenet discussion system [7] had all the key infocentric features: logical namespace and unique message identifiers, flood message propagation and caching. Peer-to-peer (P2P) networks [13, 1] identify information with its hash and allow it to propagate indiscriminately, based on user requests. Content delivery networks (CDN) employ original content URLs as identifiers, distribute the content physically over numerous servers and redirect users to the nearest replica using DNS techniques. The git [2] revision control system represents version history of a project as a directed acyclic graph of revisions, where every revision is identified with SHA-1 hash of its contents; repositories push and pull content, thus forming a network of arbitrary topology. In general, all these environments were designed for scalability when having one central data repository was not a viable option. This implicit “behind the scenes” drift towards infocentricity implies a strong demand for a generic named-data substrate that is not reflected yet in the *de jure* network architectures.

On the other hand, there are also strong counter-arguments. The landmark publication [27] suggests content-centricity as a cure for three problems of today’s networks: availability (incl. bandwidth costs), security and location-dependence. Some of those are relieved naturally, e.g. storage and bandwidth prices decrease exponentially [10] which can hardly be improved upon. Some problems, e.g. location-dependence, have an evolved set of workarounds (DNS redirections [17]) which work reasonably well. Solutions to other problems may lie outside the domain of network architectures, e.g. in economies of scale or application design.

There is yet another trend that has to be taken into account. As storage and bandwidth become cheaper at an exponential rate, the technical aspects of information propagation become less and less of an issue, compared to social and semantic aspects. For example, disseminating textual content can no longer be seen as a challenge even at its most grandiose scale. Wikipedia serves 17 million articles to 400 million unique visitors a month [5], using a very standard medium-scale infrastructure built on commodity hardware and software. Even their modest yearly budget allocates a mere 9% to Internet hosting costs [4]. There are some signs that the same post-scarcity principle might already apply to multimedia distribution as well, at least for those companies that fully enjoy economies of scale [26]. As another illustration, even cash-limited startups may not be interested [8] in saving bandwidth by using P2P these days, because server-based technology gives better controls and usage statistics that may justify further investments that will pay for the bandwidth. Projecting that into the future, eventually the value of user attention may far outweigh any technical costs.

Still, such systems as *git* focus on social challenges in the first place. Thousands of distributed developers orderly and securely weaving together the code of the Linux kernel is the real challenge addressed, while storing or downloading kernel tarballs is not. Thus, positioning infocentric

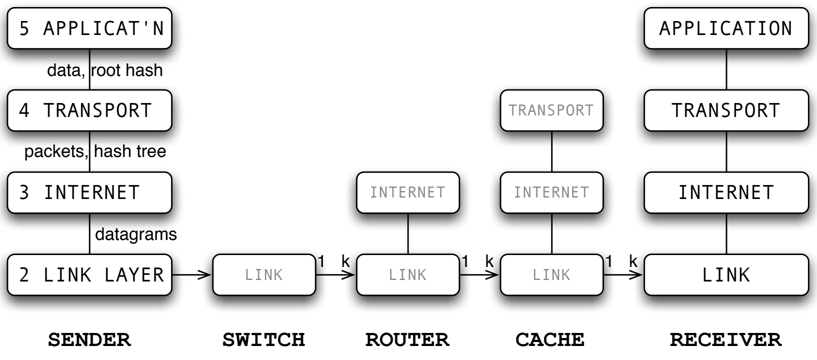

Figure 1: Infocentric protocol stack.

networking as a cost-cutter or a technical optimization is unwise. However, the possible consequences of decoupling data from storage for ubiquitous availability of data, data longevity, and general connectivity of the society look promising.

Given all the pro et contra arguments above, our point is that at this early stage, developing single-piece interdependent designs for the future Internet might not pay off. If the IPv6 story teaches us something, it is the requirement that a large-scale innovation has to be adopted piecemeal, every small step being profitable in some sense. Thus, we focus on considering a modular design with an incremental migration path, that may introduce infocentricity into a living network bit by bit, without requiring any upfront restructuring.

The *necessary* mainstay of any named-data network architecture is to employ either cryptographic hashes or signatures to enable indiscriminate caching of data in the network and the possibility of its retrieval from any available source. We prove that hashes are also *sufficient* to perform any transport function. Merkle hash trees [20] allow to identify and verify data; that perfectly enables requesting, relaying, and storing of data by any node in the network. Our variant of hash trees needs no supplementary transfer metadata; we also extend it to the case of live data streams.

We cleanly separate the network architecture into the upper "naming" part which deals with problems inherently semantic and the lower "transport" part that delivers the data, verbatim. The thin waist allows us to minimize the cross-layer interdependencies and to guarantee mutual replaceability of different naming/transport implementations. That also improves the prospects of piecemeal deployment.

The paper proceeds as follows. In Section 2 we describe our variation of the Merkle hashing scheme, its extensions, and the resulting separation of transport and internetworking layers. Section 3 addresses the implications for routing/tracking. In Section 4 we introduce a vocabulary of messages that constitutes our protocol. Section 5 describes our UDP-based implementation which confirms that the protocol may run over 3rd, 4th and 5th layers of TCP/IP stack or IP-free (over the 2nd layer). Section 6 describes what piecemeal deployment scenarios might look like. In Section 7 we address related past works. Section 8 concludes the paper.

2 The hashing scheme

Self-certifying names is the key enabler of infocentric architectures. The ability to verify data against its name allows for storage in the network, retrieval from an arbitrary location, fluid network topologies, and such. As a (packet) network may need to check data integrity piece by piece, possible options boil down to either per-packet signatures, as in [27], or Merkle hash trees. Differently from signatures, Merkle hash trees provide strict permanent identifiers of static data pieces, so we chose them as the foundation, later extending the approach to dynamic data. A hashing scheme enables the entire infocentric stack in two ways. First, it allows for a perfect application-to-transport handover. Semantically-rich and application-dependent queries are eventually converted into requests to the transport layer for particular data chunks precisely identified with hashes. Second, hashing enables infocentric internetworking, i.e. identification and relay of data chunks, data verification, and storage in the network.

At this point, we discover that data storage and data verification are highly interdependent. Namely, if a caching node does not verify data integrity, it makes cache poisoning possible. While incorrect data might not be accepted by the final recipient, an erroneous cache may form a "clot" in the network, preventing the correct data from passing through. Similarly, data verification needs storage to some degree, as Merkle hash trees need uncle hash chains to be available to verify data pieces.

That points us towards a layer separation scheme very much reminiscent of TCP/IP (see Fig. 1). Namely, there is a "hourglass waist" dumb-relay internetworking layer that only deals with separate datagrams. On top of it, there is a somewhat more intelligent transport layer that deals with entire data streams, performing verification, caching, and storage. Differently from TCP/IP, intermediate entities might transparently act as either 3rd layer routers or 4th layer "caches". In a sense, hashes replace IP addresses as endpoint identifiers, because in an infocentric network a node talks not to an endpoint at a specific location (IP address), but to a specific piece of data, whatever its physical location. Thus, a root hash suffices to open a "connection" to the network "cloud" to retrieve the data. To continue the analogy, the named-data stack is "anycast+multicast" by default. Because contents of a packet do not depend on the recipient, a request for a named data piece might be delivered to any replica and a response might be fanned out to any number of destinations at the 2nd, 3rd, and 4th layer.

In designing our modification of Merkle hash tree, we focus on smooth operation of both vertical (application to transport) and horizontal (internetworking) handovers to ensure no entities need to request third parties (as in CCN) or to retrieve additional metadata to perform their function. We ensure their operation is as simple and formalized, as possible. In Sec. 2.3, we extend our basic technique to the cases of live data streams and versioned data.

2.1 64-bit Merkle trees

We developed a variant of the Merkle hash tree scheme [20] to satisfy three key requirements: (a) per-packet data integrity checks, (b) no additional metadata and (c) suitability for live/mutable data. The general concept is to start with the root hash only, then incrementally acquire data and hashes, while verifying every single step.

Our hash tree is defined over the complete $[0, 2^{63})$ range. The reason is that we use 64-bit integers while reserving one bit. 63 bits is a good approximation of infinity at least in relation to the size of a single data chunk. All hashes are defined over aligned binary intervals named *bins*, i.e. $[i \times 2^k, (i+1) \times 2^k)$. Bins are nested, forming a strict binary tree. There are 2^{64} bins of different sizes, including one void and one root bin. We define our hash tree down to the level of 1KiB (2^{10} bytes) long bins, named *packets*. That is our expectation of the minimal MTU (link-layer maximum transfer unit). Every hash in the tree is defined the usual way, as a SHA-x hash of a concatenation of two (left and right) child SHA-x hashes. Hashes of packets are normal SHA-x hashes over 1KiB data ranges, except probably for the tail packet, which might have less than 1KiB of data. The normal recursive formula does not apply to empty bins, i.e.

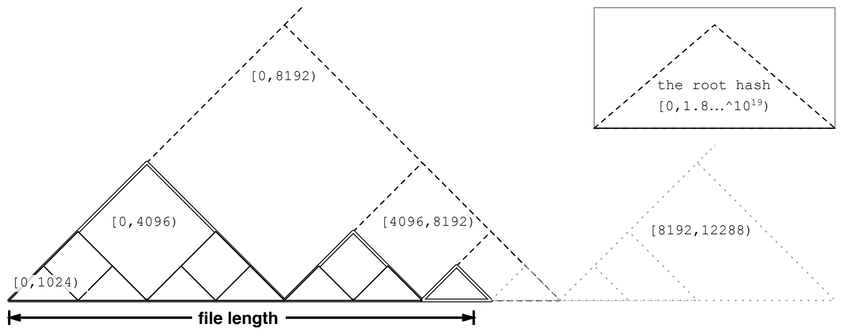

Figure 2: Bins for peak hashes are marked with double lines, filled bins with solid, incomplete with dashed, and empty with dotted lines.

bins that have no data; their hashes are set to zeros. For example, if a file is less than 8KiB long, its $[8192, 12288)$ empty bin has zero hash by definition (see Fig. 2). The *root hash* covers the entire $[0, 2^{63})$ range; this approach gives us a fixed point of reference when growing the hash tree down from the root.

So far, the hashing scheme looks quite common. The question is, how to avoid using additional transmission metadata and to extend Merkle hashes to dynamic data.

2.2 Peak hashes

The concept of peak hashes enables two cornerstone features: file size proving and unified processing of static data and live streams. Formally, *peak hashes* are hashes defined over filled bins, whose parent hashes are defined over incomplete (not filled) bins. A filled bin is a bin which does not extend past the end of the file, or, more precisely, contains no empty packets.

Practically, we use peaks to cover the data range with a logarithmic number of hashes, so each hash is defined over a "round" aligned 2^k interval. As an example, suppose a file is $l = 7162$ bytes long (see Fig. 2). That fits into seven packets ($\frac{7162}{1024} < 7$), the tail packet being 1018 bytes long. For this particular file we will have three peaks, covering $[0, 4096)$, $[4096, 6144)$ and $[6144, 7162)$ ranges. The last range might be also written as $[6144, 8192)$ because we round-up to 1KiB packet size. The number of peak hashes can not exceed $\lceil \log_2 \frac{l}{1024} \rceil$. Practically, peak hashes provide us with more convenient "reference roots" for uncle hash chains, as compared to the root hash which is 53 levels higher than the packets. Even more important is that peak hashes allow a sender to quickly *prove* the file size to a recipient who only knows the root hash. (Otherwise, file size would have to be supplied as a separate metadata piece and thus separately verified, showing up in the protocol and in the interfaces.)

Lemma 1. *Peak hashes can be checked against the root hash (if their values and ranges are known).*

Proof. (a) Any peak hash is always the left sibling (assuming axis goes left to right). Otherwise, if it is the right sibling, its left neighbor/sibling must also be defined over a filled bin, so their parent is also defined over a filled bin, which is a contradiction. (b) For the rightmost peak hash, its right sibling is zero. (c) For any peak hash, its right sibling might be calculated using peak hashes to the right and zeros for empty bins. (d) Once the right sibling of the leftmost peak hash is calculated, its parent might be calculated. (e) Once that parent is calculated, we calculate the root hash by concatenating the hash with zeros and hashing it repeatedly. □

Informally, the Lemma might be expressed as follows: peak hashes cover the entire file, so all the remaining hashes are either trivial (zeros) or might be calculated from peak hashes and zero hashes.

2.3 Live data streams

In the case of live data streams, the root hash is undefined or, more precisely, transient, as long as new data keeps coming, filling new packets to the right. Hence a transfer has to be identified with a public key instead of a root hash. Keys are more difficult to deal with than hashes, due to the reason that they have more degrees of freedom. For example, once a key is compromised, any party may rewrite a preexisting stream. While a hash might be derived directly from the data, a signature can only be verified once known, etc.

Because of those issues, we try to minimize key/signature usage by using the same peak hashes scheme as in the case of static data. Indeed, once a peak hash is defined, it never changes. Thus, we only need a logarithmic number of signatures to sign peak hashes. After that, we may deal with the same Merkle hash tree as before. We rejected another possible option of signing the root hash every time it changes. That is technically less convenient, because it requires repeated re-verification of all the peak hashes. Signing the peak hashes, on the other hand, only requires the sender to issue the newly formed peak hashes with their signatures attached. The recipient will only have to check the signature of a new peak hash and whether it matches its child hashes. Such a calculation is incremental and local.

Untill this point, we assumed that the sender emits data in “round” 1KiB long packets. What if smaller portions of data need to be committed to the network? We do not equal, but we strongly associate our “packets” with link-layer “frames”. Thus, once data is worth sending before it fills a packet, then it also needs a hash and a signature. Then, we let any packet in stream be smaller than 1KiB, which is only the case with tail packets of static files.

2.4 Conversational model

The fact that we diverge from the conversational model of TCP/IP does not mean we should not support conversation-mode traffic at all. Our primitives enable such a possibility. Indeed, data streams identified by public keys allow to *send* data to the owner of the key. Two such streams identified by a pair of public keys may allow a TCP-like bidirectional connection. At each side of a connection, a sender encrypts the data using the recipient’s public key and signs it using own private key. Correspondingly, the receiver checks signatures using the sender’s public key and decrypts it using his own private key.

3 Routing and tracking

The general infocentric paradigm of “talking to hashes or keys” instead of “talking to IP addresses” has both upsides and downsides. On the upside, the transport becomes oblivious to network topology; the topology may be fluid, multihoming is much easier, and there is no need to rely on physical infrastructure for basic security. On the downside, flat-label routing schemes tend to employ per-stream routing table entries [27] or DHT-inspired topologies with somewhat unoptimal path lengths [19]. Routing table sizes being dependent on the data flow is a tremendous challenge, especially at the Internet scale. Although this area definitely needs further research, when refocusing from the Internet scale to smaller niches, we might see numerous examples of similar problems successfully tackled. For example, BitTorrent DHT successfully handles hash-to-IP resolving for approx. 3mln swarms and 30mln simultaneous peers under the conditions of high churn and extreme unreliability of nodes. Recent measurements show that a practical estimate for DHT request latency is on the order of 200ms [22]. Similar problems are solved by content delivery networks and big-data providers, with a somewhat higher number of users and content pieces, albeit using nodes of high availability. From a theoretical standpoint, this problem is perfectly parallelizable as handling one hash does not depend on another. Generalizing that, we assume the existence of a mapping scheme, in principle resembling ARP [21], that maps hashes, as content identifiers, to lists of lower-level (e.g. IPv4 or MAC) addresses.

4 Message vocabulary

When designing our message vocabulary we minimized the assumptions about the underlying link layer. We call it a "vocabulary" and not a "protocol" because we do not specify particular serialization or encapsulation schemes, neither do we specify message exchange patterns beyond the very basic requirements. Such a "vocabulary" might be instantiated as an infocentric transport/network "protocol" (as in Fig. 1) running over virtually any medium. As this component essentially forms the "waist" of our infocentric hourglass model, we focus on simplicity and formalization of the function, avoiding loose semantics.

A DATA message simply carries the data. As it must work over diverse mediums, such as streams, datagrams, or frames, we only specify that a DATA message must carry a bin of data (i.e. 1KiB, 2KiB, 4KiB, etc). Allowing to send arbitrary intervals will introduce excessive freedom/looseness, starting from the fact this would make it difficult to check the data against the hash tree at once. Processing of uniform chunks or multiples of chunks is a robust principle underlying data networks, construction, and transportation.

A HASH message carries one hash from the hash tree. We ensure it is possible to supply the recipient with parts of the hash tree incrementally, to allow for an amortized and local verification of data. We assume that the sender has to ensure the receiver has every hash needed to verify the incoming data immediately. In case the underlying (link) layer employs datagrams/ frames, we advocate the principle of *atomic* datagrams. Namely, that every piece of data must be verified once received and either accepted or dropped.

As we allow for the possibility of data retrieval from multiple sources in parallel, the vocabulary necessarily employs both positive (ACK) and negative acknowledgements, i.e. requests (REQ). Acknowledgements follow the logic of hash trees as well; the data has to be acknowledged in bins.

We want to be independent of the underlying link/ transport/ application protocols and still conduct several transfers in parallel. Thus we define "channels", i.e. ongoing transfers, each dealing with a single data piece identified by either a hash or a public key. Means of conveying channels depend entirely on the underlying medium; that may be either a datagram header in case of UDP or a GET query field in case of HTTP.

5 The UDP implementation

We prove the feasibility of our design by implementing a multiparty transport protocol named *swift* [3] layered on top of UDP. Because of the specifics of the protocol, successful functioning over UDP also proves a possibility of running over other unreliable-datagram mediums, such as IP or Ethernet. A possible IP implementation differs by the absence of UDP port multiplexing. Thus, an over-IP protocol would have to run in the kernel (*swift* runs in user space). As infocentric data routing is based on hashes, port numbers are not necessary to forward data to interested applications. Similarly, implementing the protocol over raw Ethernet removes IP

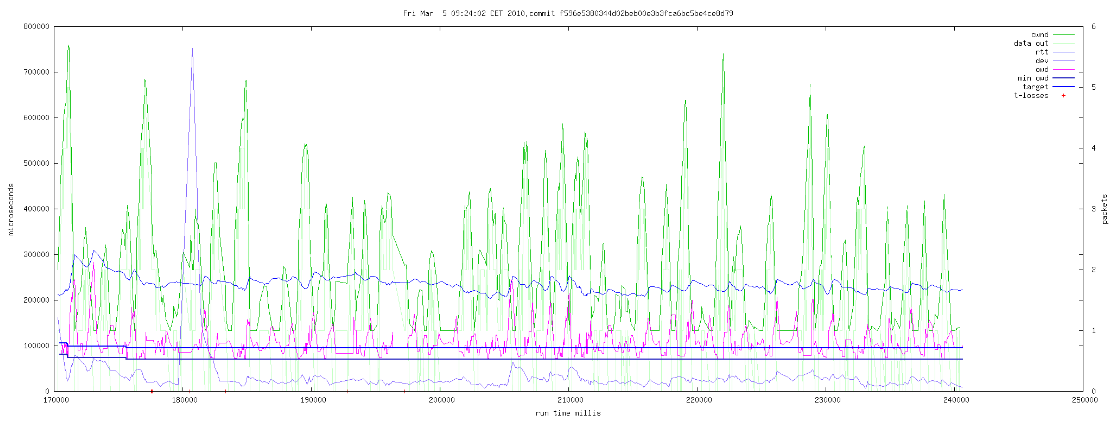

Figure 3: LEDBAT congestion control avoids losses.

addresses. Again, IP addresses per se are not necessary for running infocentric data transfers. (An experimental branch of *swift* runs over Ethernet.) The UDP implementation also trivially proves feasibility over reliable streams (e.g. TCP).

The detailed design of our *swift* protocol is described in an IETF draft [14]. The protocol is a direct implementation of our vocabulary with some additions and extensions. Messages are serialized as fixed-width fields starting with a single-byte message type field, followed by fixed-width payload fields, such as bin numbers, data, hashes and such. Messages are packed into UDP datagrams. Datagram processing is event-driven, fully implementing the atomic datagram concept. Namely, every datagram is either immediately committed to storage or immediately dropped; there are no buffer-reassembly mechanics. Our implementation of *swift* employs the LEDBAT [23] scavenger congestion control algorithm, which allows streams to run virtually lossless under normal conditions (see Fig. 3).

Swift has certain novel features, compared to well-known protocols, such as TCP or BitTorrent. Predictably, it faced new challenges, mostly caused by (1) per-packet integrity checks, i.e. hashing and (2) stripy transfer progress state, caused by data coming out of order.

To guarantee that a receiver can verify every packet, the sender has to prepend it with the missing uncle hashes. Luckily, the receiver builds the hash tree incrementally, so in a perfect network with no data loss, every packet of data needs one hash on average. More precisely, every *even* packet needs a hash for its sibling, every fourth also needs a hash for its uncle, every eighth also needs a hash for its parent's uncle, etc, thus the average is 1. In practice, some packets are lost, so a prudent sender overprovisions hashes to compensate for possible loss. Thus, the actual traffic overhead of hashes is somewhat above the perfect value of 2% (assuming 20 byte hashes for 1024 byte packet). Another technical problem, SHA-1 hash calculations are CPU-intensive, and hashing data in smaller 1KiB portions was known to increase the (already high) CPU consumption at least twofold. Investigation of the issue has shown it was rather OpenSSL-specific and that the git [2] SHA-1 implementation – which is specifically optimized for hashing smaller chunks of data – does not have that issue.

The protocol needs to keep more state on the transfer progress, as data might arrive totally out of sequence – mostly because data is often delivered from different sources in parallel and those sources might have stripy state as well. In various peer-to-peer protocols, that problem is often resolved by subdividing the data into "pieces" and then pieces into "chunks". Instead, we adopted a generic compressed-bitmap data structure named “binmap” [15], a hybrid of a bitmap and a binary tree, which allows to track data at an arbitrary scale, starting from a single packet. That state also has to be communicated over the wire, using unreliable datagrams. Acknowledging and requesting data in bins provides necessary compression, as continuous chunks of data are acknowledged with a logarithmic number of messages. Also, it provides soft protection against datagram loss by means of redundancy, as the same received packet ideally gets acknowledged repeatedly, as part of larger and larger bins, without increasing the overhead.

As a result, we have implemented an infocentric transport protocol over unreliable datagrams, without supporting the intermediate abstraction of an ordered data stream. The libswift implementation *confirms* that the proposed vocabulary is universal and functional enough to implement infocentric internetwork and transport protocols running over diverse underlying link technologies, providing a simple “data for a hash” interface to the upper application layer and thus acting as a perfect hourglass “waist”.

6 Scenarios

In this section, we provide several example scenarios for piecemeal deployment of the architecture in question, mostly to showcase its flexibility, without claiming usefulness for any particular purpose.

6.1 ICN over HTTP

A very limited subset of the described model, namely the hashing scheme, may be used to orderly identify and cache static data pieces, e.g. multimedia files. The only requirement is a URL syntax convention. File names based on Base64-serialization of SHA-1 Merkle root hash might work well: `flower=qhbdHl4A7knaJVS73gGldT7Rq4e.jpeg`. Such a data piece might be assigned an indefinite caching time.

6.2 Peer-to-peer Web

Instead of putting an IPv4 address on DNS, a web site may put a root hash, disguised as a server name, e.g. `4FlSei8vJe4aA2Wd9Qef6UIOCme.icngw.net`. Once a user
makes an HTTP request, a local transparent HTTP proxy intercepts it. It looks up the hash on DHT to find peers that have the data. Then the proxy retrieves the data and delivers it to the user by HTTP. For backward compatibility, the name points at a public gateway `icngw.net` that may deliver the page to regular HTTP clients. Essentially, the logic for a proxy and a public gateway is the same.

6.3 Flat storage

An improvised storage keeps data pieces (e.g. multimedia files) identified by hashes spread to several servers sharing the same Ethernet switch. Data is stored on raw hard drives using flat hash-indexed allocation tables. Servers receive HTTP requests from outside users interested in data pieces. To locate a piece, they issue local broadcast requests. In case no one has it, they retrieve the piece from external sources, send it to the user and cache it for later. In case other servers have a copy, they redirect the HTTP request to that server.

7 Related Work

There are numerous proposals investigating novel network architectures based on the infocentric paradigm. Most studies try to embed the infocentric concepts deeper into the network architecture through augmentation of interdomain routing, separation of applications from transport, and/or by introducing variations of (secure) content naming schemes.

TRIAD [12] uses URL-like names to identify the content, where DNS part of the URL is used by content routers for next-hop routing. The closest replica is found through this name-based forwarding and content retrieval is done through standard HTTP/TCP.

Similarly, the content-centric networking (CCNx) model proposed by Van Jacobson et al [27] relies on URL-like identifiers to name content. Names have a hierarchical structure based on which forwarding, routing, and storage are carried out. CCNx forwards Interest messages along paths where content may be located. Nodes possessing requested content return it in form of Data messages along the reverse path. Content may be cached by any router in the path.

Data-Oriented Network Architecture – DONA [25] is another CCN approach where content is identified by flat, self-certifying labels instead. Content is registered beforehand with a tree of resolution handlers (RHs) and can be retrieved through the *find* primitive and standard IP routing. Content can only be offered by the principal or authorized entities.

NetInf naming scheme [9] is similar to DONA [25] in terms of using flat names and a publish/subscribe API, but it additionally incorporates a non-trivial set of security options. Content (videos, web pages, photos, etc.) is represented as *information objects* (IOs), where each IO is identified by a flat, self-certifying and location-independent *ID*. Sources publish IOs beforehand and requesters subscribe for IOs they are interested in. An ID is dynamically bound to several network locators where copies of IO are stored, but the real binding to network locations is done externally by a network resolution service (NRS). Unlike in DONA, NetInf IDs remain intact even if the owner of content changes, thus assuring name persistence.

In [28], Wong et al propose a solution which ensures backwards compatibility with the current DNS naming system. In this proposal, human-readable (URIs) labels are used to name content but a secure mapping from URIs to cryptographic identifiers is provided. Finally, DNS is used to resolve such identifiers into network labels.

Some proposals make use of distributed hash tables (DHTs) to route name-based queries. ROFL [19] (Routing on Flat Labels) routes on flat identifiers and routing occurs as in Chord [16]. SEATTLE [11] uses a one-hop DHT to build a directory service which does host address resolution and service discovery. As in DONA, where content must be registered with RHs before it can be retrieved, DHT-based proposals must also register the location of content in advance.

Popa et al [18] argue that HTTP is a content-centric protocol already providing many of the features proposed in CCN/ICN architectures. The proposal leverages existing deployed HTTP infrastructure such as CDNs, HTTP proxies and caches. HTTP makes use of content names which it binds to DNS names in form of URLs. Proxies along the way can cache content or even redirect requests to closest servers. This paper implements an extension S-GET which imitates the publish/subscribe API proposed in most CCN architectures. In S-GET, a user subscribes for a URI (content) by storing an S-GET request to the closest server. On the other hand, the content provider sends all updates for that URI – to the subscriber – using HTTP PUTs.

Braun et al [24] propose an extension of CCN into a service-centric networking (SCN) scheme which would support a variety of services such as audio/video streaming, recording and processing of images/videos, file storage, location-based services and more. The paper proposes to extend the naming scheme – used for content – to name services as well. SCN leverages Interest and Data messages used in CCN: client sends an Interest message which specifies the service; the underlying CCN infrastructure performs named-based routing on the message; results from the executed service are returned back to the client in Data messages.

8 Conclusion

In this paper, we considered a modular design for infocentric architectures with prospects of incremental deployment. We took the very essential and basic infocentric primitives, such as self-certifying names, and followed their implications on the network architecture. We focused on the transport layer, leaving routing, naming, and the application layer out of scope and making no strong assumptions about them. As a result, we came to a natural separation of infocentric transport and network layers, somewhat reminiscent of the TCP/IP, but with many new interesting abilities and possibilities. Effectively, we designed an infocentric "hourglass waist" – based on Merkle hash trees – and a corresponding message vocabulary. We have confirmed the feasibility of our core algorithms and data structures by developing and testing a prototype protocol. Instead of devising new "future Internet architectures", we considered options and scenarios for piecemeal adoption of infocentric infrastructure in existing networks.

References

[1] eMule project. http://www.emule-project.net/. 4
[2] The git source code management system. `http://git-scm.com/`. 4, 10
[3] libswift homepage. `http://libswift.org/`. 9
[4] Wikimedia Foundation annual plan 2010-2011. `http://wikimediafoundation.org`. 4
[5] Wikistats: Wikimedia statistics. `http://stats.wikimedia.org/`. 4
[6] Cisco visual networking index: Usage study. `http://cisco.com`, October 2010. 4
[7] R. Allbery and C. Lindsey. RFC 5537: Netnews architecture and protocols. 4
[8] Arvid Norberg of BitTorrent, Inc. private conversation. 4
[9] C. Dannewitz et al. Secure naming for a network of information. In *INFOCOM IEEE Conference on Computer Communications Workshops*, 2010. 12
[10] C. Labovitz et al. ATLAS internet observatory 2009 annual report. NANOG47 presentation. 4
[11] Changhoon Kim et al. Floodless in SEATTLE: A scalable ethernet architecture for large enterprises. In *SIGCOMM*, 2008. 12
[12] D. Cheriton and M. Gritter. TRIAD: A new next-generation internet architecture, 2000. 12
[13] B. Cohen. Incentives build robustness in BitTorrent, 2003. 4
[14] V. Grishchenko. The generic multiparty transport protocol (swift). `draft-grishchenko-ppsp-swift-02.txt`. 10
[15] V. Grishchenko and J. Pouwelse. Binmaps: hybridizing bitmaps and binary trees. PDS Technical Report PDS-2011-005. 11
[16] I. Stoica et al. Chord: a scalable peer-to-peer lookup protocol for internet applications. *IEEE/ACM Trans. Netw.*, 2003. 12
[17] J. Dilley et al. Globally distributed content delivery. `http://akamai.com/`. 4
[18] L. Popa et al. HTTP as the narrow waist of the future Internet. In *Proceedings of the Ninth ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010. 12
[19] M. Caesar et al. ROFL: routing on flat labels. In *SIGCOMM*, 2006. 8, 12
[20] R. C. Merkle. A digital signature based on a conventional encryption function. In *CRYPTO*, 1987. 5, 6
[21] D. C. Plummer. Rfc826: An ethernet address resolution protocol, 1982. 9
[22] R. Jimenez et al. DHT measurements. manuscript in preparation. 9
[23] S. Shalunov et al. Low extra delay background transport (LEDBAT). `draft-ietf-ledbat-congestion-04.txt`. 10
[24] T. Braun et al. Service-centric networking. In *Fourth Intl. Workshop on the Network of the Future*, 2011. 12
[25] T. Koponen et al. A data-oriented (and beyond) network architecture. In *SIGCOMM*, 2007. 4, 12

[26] Unnamed Google employee. private conversation. 4

[27] V. Jacobson et al. Networking named content. In *CoNEXT*, 2009. 4, 6, 8, 12

[28] W. Wong and P. Nikander. Secure naming in information-centric networks. In *Proceedings of the Re-Architecting the Internet Workshop*, 2010. 12