Packet-level Caching for Information-centric Networking

Somaya Arianfar, Pekka Nikander
Ericsson Research, NomadicLab
firstname.secondname@ericsson.com

Jörg Ott
TKK, Comnet
jo@netlab.tkk.fi

June 4, 2010

Abstract

Within the current Internet architecture, adding fully efficient in-network packet caches is not easily doable. Packets are defined only in the context of transport level flows and are not independently addressable. In this paper, we introduce a new model that makes fully functional network-level packet caches possible. We give early evaluation results on the feasibility and usability of in-router packet caches, within the new model.

Chapter 1

Introduction

In the Internet, a large amount of content is transferred repeatedly [15]. Most of the time, the content is retransmitted from the source to serve different requests coming across the network. The efforts to reduce the amount of repeated traffic can be roughly divided into two categories: application level caches, including web caches [30] and Content Delivery Networks (CDNs, cf. e.g. [1]), and application-independent caches, which can be often found in the so-called WAN optimization products [14]. Today, these approaches together offer significant improvements to the network performance by caching some of the content.

Besides packet and chunk-level caches used in WAN optimization, in-network packet-level caches are also (re)gaining academic interest [4, 6]. It has become economically and technologically feasible for network devices to store large amounts of data [13]. Using intelligent and targeted caching at selected nodes, it is possible to reduce the network load, shorten the experienced latency, and avoid hot spots [4, 24].

Typically, in-network packet-level caches are independent of the end-to-end transport logic and therefore introduce less overhead at the caching points than the upper-layer caches. However, in spite of their simplicity and effectiveness, the packet-level caches are today used only for reducing link load, while upper level caches can also help with other inefficiencies, i.e., high latency and server load [1, 10, 30].

In this paper, we show how information-centric networks can enable fully functional packet-level caches and use them as general network components. We look at some of the inefficiencies that can be managed with packet caches, specifically from the transport layer point of view. We finally show some early evaluation results (on the feasibility) of our model.

Chapter 2

Background

A main motivating factor for our work are transport layer inefficiencies, determining if wide-spread in-network packet caching could contribute to eliminating most retransmissions. In the present evolving environment, the commonly used reliable transport protocols are becoming increasingly inefficient and unstable [18], partially due to them relying heavily on end-to-end feedback loops. In particular, most of today's reliable end-to-end protocols require retransmissions all the way from the source. This, combined with the typically extensive queuing at congestion points [28], leads to degraded performance [18] and to relative waste of costly resources, such as using large amounts of DRAM only for buffering.

There have been some efforts to address the latter problem, both by combining layer 2 and 3 feedback [3] and by changing the transports, cf. e.g. XCP [18]. There have also been studies on using in-router infinite caches to improve the efficiency of specific packets [20, 29] and for application-independent schemes at the packet level to detect and reduce redundant traffic [26]. While some sources of retransmissions, such as lossy radio links or transient disconnections, are addressed locally in some new networking architectures (e.g. DTNs [12], cache and forward [22]), the buffering problems remain. In particular, other than the proprietary fingerprinting-based caching in WAN accelerators [14], there is no widely deployed network-layer functionality that could shortcut retransmissions from the source.

In this paper, we propose using router memory as a cache rather than as a buffer. We rely on the fact that, once a packet has been written to memory, it remains there until it is overwritten by another packet. However, allowing buffers to be used as caches requires some changes to the network architecture; we propose one particular approach, borrowing ideas from information-centric networking [16, 27]. Our work differs from [3, 18] by explicitly using the memory in routers as a cache, not as a buffer, and from [20, 29] by justifying the feasibility of the model as a general network optimization.

Chapter 3

Conceptual overview

Designing fully effective packet level caches in routers requires packets to be reusable. As prerequisites, packets need persistent global identifiers that are valid independent of individual transport connections and suitable retrieval mechanisms need to be in place. The architecture needs to address common problems (e.g. support for versioning and for discrete or streamed contents) and the in-router cache needs to be efficient to operate at line speed. Hence, we revisit the way to look at the packets in the networks and how they are treated.

3.1 Content-centric networking

In 2006, Van Jacobson introduced the concept of content-centric networking [16] at the Stanford Clean Slate seminar. In the PSIRP [27], content-centric networking is used as a base for defining a new architecture. Identifiers are defined in an information-centric manner. Content retrieval and inter-networking are based on topic-based publish/subscribe [11] instead of the present send/ receive paradigm. The network has the responsibility of matching subscriptions to publications and of delivering the content.

A particular way of implementing the PSIRP architecture is called RTFM [25]. It has an explicit rendezvous function, to allow subscriptions and publications to meet within the system. An overall view of the RTFM architecture is depicted in Fig. 3.1. Subscription matching (rendezvous), routing (topology), forwarding, and caching are performed in a recursive manner at different abstraction

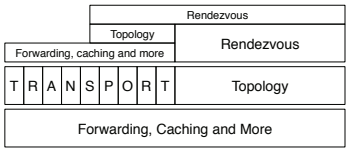

Figure 3.1: The RTFM architecture

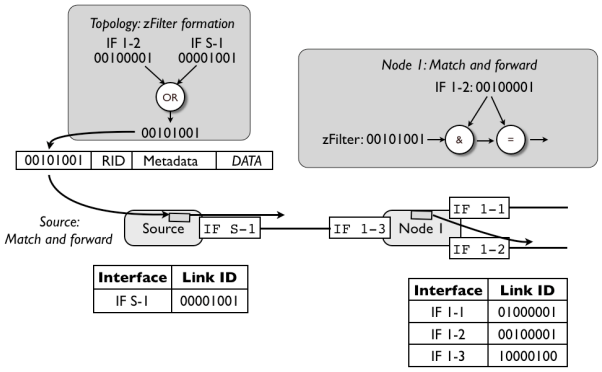

Figure 3.2: An example of the packets and forwarding, adapted from [17]

layers, starting from high level document objects down to the level of individual packets.

3.2 Identifiers

We use the RTFM model as the basis for our packet identifiers. As depicted in Fig. 3.2, each packet has two identifiers: a Forwarding identifier (FId) and a Rendezvous identifier (RId). The FId is a networking layer identifier, used to forward the packet through network. The FId can be considered as an equivalent to the src/dst address pair in IP packets. In [17], each FId explicitly lists all the unidirectional links along a transmission path (or a tree), packed in an in-packet Bloom filter. The forwarding decisions are based on each node checking if one or more of its outgoing links are included in the in-packet Bloom filter; the packet gets forwarded along those links. This operation is shown in Figure 3.2.

Each packet also contains its own Rendezvous identifiers (RId). Multiple RIds can be used to determine how the packet fits into larger data abstractions, such as documents or streams. For instance, each logical document may have its own high level RId, different versions of that document their own RIds, and each part of each version again its own RId. At the packet level, the packet-level RIds are used to identify, access, and retrieve packet-sized content in the network. A packet-level RId can be, for instance, a cryptographic fingerprint computed over the logical, invariant content of the packet. Alternatively, for real-time streaming, the packet-level RIds may be generated by a hash chain function from a seed identifier (higher level RId). For the purpose of this paper, we assume packet-level RIds to be 128-bits-long hashed identifiers over the packet's data content. Consequently, the RIds themselves can act both as a sequence numbers[1] and as secure content identifiers.

[1] Assuming the receiver knows which RIds should arrive next; see below.

3.3 Packet retrieval

Retrieving packets from a cache requires subscribers to know the packet RIds beforehand. In our design, before the actual transfer of any data begins, the receiver (subscriber) obtains a list of packet RIds that make up the object it desires. It then requests each of the packets (logically) separately; in practice, any node may optimistically push packets whenever there is free capacity over its outgoing links.

For this, the source constructs a meta-data object, containing the packet-RIds of the individual packets, and publishes it toward the receiver. If there is an update to an object, the source will update the meta-data object, republish it, triggering the receiver to get the new version, allowing it to construct the new version of the object, and even to republish it later.

With this design, it would be sufficient for all packets to contain just the FId and the packet-RId. The meta-data object, would also contain the higher level "object" RId. The "object" RId allows the receiver to distinguish different versions.

3.4 In-packet metadata

Unlike the IP model, where packets in the network are similar to each other, we need some distinction between different packets. Caching points also need to distinguish between data packets (to be cached) and (re)transmission request packets (to be processed). To address this issue, we introduce in-packet meta-data. This metadata, included in each packet, is distinct from the higher-level meta-data objects. It contains information about the type of the packet and its cacheability lifetime. In-network caching points make the caching and retrieval decisions based on the included meta information.

Chapter 4

Design

In our design, caching becomes part of the router functionality in a similar way queuing is today. In the following, we present our design in detail, specifically taking efficiency into account.

4.1 Cache structure

Our router cache contains two main components: a packet store and an index table to access the store. Packets are assigned to different cells in the packet store. The index table contains information to find the packet in the packet store. The packet store is large and can be kept in the router's DRAM The size of index table is smaller, thus it can be kept in SRAM.

A packet store (DRAM) cell address can be pre-defined based on the index position or it can be calculated based on available free space at the time of insertion. We currently use pre-defined fixed size entries in the packet store, similar to other packet level caches, see e.g. [4].

4.2 Operations

4.2.1 Insertion

In our base model, almost all the packets entering the router are indexed and cached. Later on, we extend this model with random fractional caching, where each packet has a pre-set probability of getting indexed and cached. It would also be possible to apply explicit co-operation between routers, similar to [6], but we leave that for future work. Indexing is done independent of the FIds and, as a result, packets can be retrieved independent of the original path.

Each entry in the index table contains at least the *packet RId* and some *access history*. The overall local cache system is depicted in Fig. 4.1. A hash of an RId, $H(RId)$, defines the position of the index entry in the index table. Packet itself is then written down in the assigned location in the DRAM packet

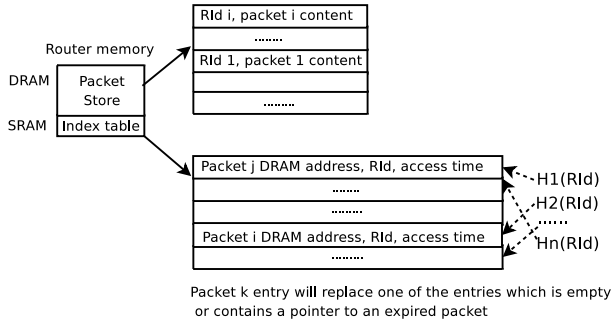

Figure 4.1: Structure of packet store and index table in router's memory

store and its index is updated in the index table. Indexing packets in the cache can be done with different methods based on the tradeoffs between space and speed. To achieve a good speed, we use universal hash functions to index the data in O(1) (cf. e.g. [8]). They distribute the packets randomly in the memory, so there is no FIFO tail competition similar to what exists between TCP flows today [9]. To increase the probability of finding free space, we employ a number of parallel hash functions, H_i, preferring a free location if available.

4.2.2 Deletion

We follow a lazy deletion model, meaning that a stored packet is flushed out upon its lifetime expiration only if a new packet arrives and needs to be inserted in that location. In this way, the new packet's index replaces the expired packet's index in the index table and the new packet itself overwrites the existing packet in the packet store.

4.2.3 Look up

Upon receiving a request to retransmit a packet (which can be compared to receiving NAKs in the current Internet model), the router looks up the requested packet's RId in its index table. The router does it by hashing the RId of the requested packet to find out the possible locations in the index table. It retrieves the entries from the index table and matches the RIds in each retrieved entry against the requested RId. If a match is found, the packet itself is retrieved from the DRAM and forwarded to the requester, using the FId included in the request. The index table allows the router to schedule the sending of the requested packet directly out of the DRAM, making it almost comparable with systems without cache lookups, as our lookup time is independent of the number of packets in the cache.

4.3 Locating and forwarding data

As mentioned earlier, the initial matching between the location of data and a new subscriber is performed by the rendezvous system. A FId is computed by the topology layer, and the meta-data object is pushed to the subscriber. After this initial phase, the subscriber knows both the RIds of the actual data and a (time limited) FId of the forward path; we assume that the FId can be used in the reverse direction for packet requests, bypassing the rendezvous. Hence, the subscriber is both able to ask for new data and to send retransmission request for cached packets. A packet request can be served by the next upstream router on the path, if has the packet in its cache, or it can be handed over to the next hop upstream router, to the source, to other receivers of same data (if known), or to any combination of them.

4.4 Caching and queuing

In today's non-caching routers, assuming congestion, each received packet is buffered in the DRAM and a pointer to it is enqueued in SRAM, for scheduling. The same simple queuing can still be used in our system, if needed, assuming that the queue length needed to absorb congestion is shorter than the desired caching time. (In the cases we consider, the typical queuing time should remain well below one second while the cache should hold a few seconds worth of packets.)

Depending the details of the cache replacement algorithm, there may be a small conflict between the caching and queuing requirements. Consider a new packet that needs to be queued but happens to hash only to occupied locations (typically by *cached* packets). Here we have two options. Firstly, we can forcedly delete one of the already cached packets, freeing the location for the new packet. Secondly, as collisions are rare, we can use a small amount CAM to store the indices for colliding high-priority packets. The former method is especially usable in the case of random fractional caching (see below), while the latter is better if we aim for full traffic caching. In the unlikely event that all storage locations are occupied by other *queued* packets, this results either in a packet loss by dropping the incoming or overwriting an already queued packet, or requires some extra space in CAM as described above.

4.5 Transport and cache interactions

A router cache can be used to help both push- and pull-based transport protocols. In the packet-level push mode, data packets are pushed out by the source. The sinks send (re)transmission requests only when they discover that some packets were lost or when when they reconnect after a short disconnection period. Such requests can also come, in a more pull-like mode, from a new node that has just got a new meta-data object. Having caches on paths leading to lossy or congested links can save considerable retransmission resources,

especially in the case there are multiple subscribers generating identical retransmission requests.

In the packet-level pull mode, where the sources do not actively push any data packets, the routers can be more active. Instead of relying on the source to push or the sink to pull the data, a router can actively pull data from the source, adjusting the pull rate to its cache capacity and other metrics. The details of the pull mode remain as future work.

Chapter 5

Evaluation

To get some indication of the feasibility of our approach in routers, with or without using RTFM, we have run a number of ns-3 simulations and made some estimates of the component and energy costs. While the results of this early evaluation are far from conclusive, they indicate that wide-spread in-network caching could reduce the network load considerably at little or no additional cost compared to today's routers.

5.1 Simulation results

As a quick sanity check, we simulated the collision rate of our ns-3 cache implementation with full caching. We used universal hash functions [8] for indexing, as they are fast and easy enough to be implemented in hardware (see e.g. [19]). We alternated between 7 and 8 parallel hashes over the packet RId; a matching empty/ expired location was replaced. The case when all matching locations contained non-expired packets was considered a collision. Each test was ran for a few seconds, with the constant lifetime of $1s$, with random RIds.

We used a linear path of 8 routers, initially each trying to cache *all* packets on a set of flows totaling 8000 packets per second. Table 5.1 shows the average percentage of colliding packets at each router. The results indicate that caching can be quite efficient with a number of hash functions and enough space.

Next, we tested our model to study its efficiency with retransmission requests in the case of full caching. In this straightforward test, retransmission requests arrived within the caching lifetime; this appears to be a valid assumption, considering the lifetime of a few seconds for cached packets [5]. Requests were generated randomly by end hosts connected to the last hop router in the path. Our source pushed packets at a uniform rate, and kept copies of them all in its internal cache.

To study the effect of random fractional caching, we varied the percentage of packets each router caches, choosing the cached packets randomly. The results, shown in Fig. 5.1, indicate that the overall mean efficiency can be quite high,

Packets per second (#)	8000	8000	8000	8000
Cells in index table (#)	8000	8000	16000	16000
Hash functions (#)	7	8	7	8
Avg. collision rate	26%	24%	1%	0.3%

Table 5.1: local collision rate

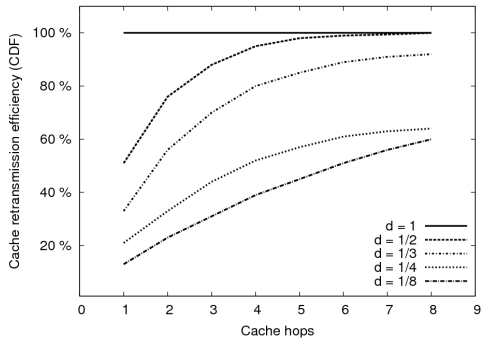

Figure 5.1: In-network cache efficiency for different caching probabilities

even with partial caching at each router. When reducing the caching fraction, the number of routers needed for a given coverage increases; with full caching, the first 2 caches cover all the requests, while with 50% random caching, 4 caches achieve 99% hit rate. For a low caching fraction (e.g., 1/8), the hit rate across 8 hops converges to around 60%.

5.2 Feasibility considerations

5.2.1 Capacity

Both early (late 1990s) work on web caches, while they still were effective [30], and recent work on the time between the first and the last match in packet level caches [5], indicate that a few seconds of packet caching leads to major benefits. The results in [5] explicitly suggest that around 50% of the potential retransmission savings can be achieved within 10 seconds. Considering 10 Gb/s line speed, a suitable amount of in-router packet memory would need be 100 G bits, to keep all the traffic passing through an interface for 10 seconds, costing some $200–300 with today's prices.

Assuming a large packet size of 1500×8 bits, potentially typical for the kind of content-centric networking we are considering, indexing 100 G bits of data would need some 9 million index entries in SRAM. We can estimate that each entry might take around 160 bits: 128 bits for the RId plus a few bits for state.

Considering that commercial SRAM chips are commonly available 36 Mb (512k×72) configurations, with the assumed price of $5–7 per chip, the required 1.5 Gbits of SRAM for indexing large packets would cost some $200–280. Hence, our proposed caching system would cost ≈$500 per line interface for a $10s$ long

full cache Given that multi-port line cards for commercial routers have typical per-port cost in the range of $1500–2500 per port, our ballpark figure appears reasonable.

5.2.2 Processing throughput and memory latency

With a suitable ASIC or FPGA design, computing the hash values, examining the resulting SRAM index locations, picking one of them, and writing the new index takes at most a few clock cycles. The latency associated with writing or reading a packet to/from the DRAM may take a large number of cycles, though, especially if slower and less power hungry DRAM is used. For example, this time assuming the worst case of short 40 byte packets, OC-192 (10 Gb/s) and OC-768 (40 Gb/s) give packet inter-arrival times of $32ns$ and $8ns$, respectively. Today, the typical DRAM random access time is around $50ns$. This means that it is quite feasible to randomly access a memory bank once for every 2 packets for OC-192 and once for every 7 packets in OC-768. In addition to using parallel banks, random fractional caching may be used to make sure that the DRAM latency does not become a bottleneck.

5.2.3 Energy consumption

Of course, power consumption can become a limiting factor. Usually SRAM circuits consume significantly less power than typical DRAM circuits, allowing us to assume moderately fast SRAMs. Furthermore, assuming the DRAM banks are (almost) constantly written, the need for refresh cycles is reduced significantly. This makes DRAM almost as power efficient as SRAM. Our system not requiring fast memory but multiple banks of slower memory helps to keep the total energy consumption reasonable. Given that a commercially available 1GB DRAM module uses around 1W of power when active all the time [2], 12 GB (100 Gb) would use around 100 kWh per year. Assuming a high power cost of $0.20/kWh, the annual cost would be in the order of $20. Furthermore, the resulting thermal load is likely to be relatively small compared to that of the rest of the system.

In our system, since caching is not done as a hard reliability component in the network but rather an opportunistic optimization, doing partial caching can always reduce the costs on single routers and still keep the overall efficiency high, though the memory should be used in a proper way for queuing, too.

5.3 Discussion

The results above indicate that given today's component capacities and prices it would be quite feasible to cache a few seconds worth of traffic at each router, with a fractional increase in the component cost and marginal increase in the operational costs. Using coordinated caching [6] or fractional random caching, the cache can be extended considerably, up to a few minutes. The results

in [5] indicated that with a $100s$ long cache about 60% of repeated requests could be served from the cache, while with a $1000s$ (17 minutes) long cache the corresponding figure was 80%.

Considering router memory as explicit packet caches appears to have a potentially drastic effect on future transport protocols. First, given our design, a router would be able to distinguish data and meta-data objects, giving the meta-data objects priority. Second, the explicit flow identifiers (FIds) would allow the routers to easily handle each flow separately (cf. [23]) avoiding the usual latency problems associated with large buffers. Especially, each router could explicitly send flow-specific congestion feedback upstream (cf. [3]). This results in a system with serial cascade feedback loops [7], potentially increasing the overall stability of the system.

A 1000 seconds long distributed cache would contain about 1.25 TB of data; we can imagine it easily absorbing bursts up to a few MBs. This might make the TCP slow start and similar mechanisms unnecessary, as the cache would absorb data well until the feedback from the congested router would hit the source. The few packets potentially missed due to collisions could later be recovered from the source.

For multicast, the design easily adopts to a situation where some receivers are behind links with fluctuating capacity. The router next to the problematic link can prioritize meta-data, cache some content locally, and pull the rest from upstream and sibling branches as soon as the congestion clears.

5.4 Conclusion

In this paper, we have suggested instrumenting routers to act as (probabilistic) packet caches with minimal overhead while maintaining their queue-based forwarding functionality. Using additional memory for caches larger than typical queues is technically motivated by the existing redundancy at the packet level [5] and economically supported by the faster decrease in memory prices compared to bandwidth and latency [13, 21].

We have shown that fully functional packet caches are feasible in today's networking environment and that they could become part of routers. Placing caching functionality in routers avoids the need to determine optimal locations for caches inside the network and indirections since they are implicitly on every data path. As our caching operates opportunistically, the available memory could be increased gradually and the caching probability could be tuned to meet the requirements of different locations in the network topology. Finally, there is no need for coordination between different service providers or even different routers within a single provider, except for choosing disjoint hash functions

While the approach looks promising, further evaluation and an actual implementation are needed to gain further insights into its suitability. Our next steps include devising transport and application protocols along the lines outlined above to determine how well our design meets their needs.

Bibliography

[1] Akamai technologies. http://www.akamai.com/.

[2] Micron system power calculator. http://www.micron.com/support/part_info/powercalc.aspx.

[3] M. Alizadeh, B. Atikoglu, A. Kabbani, A. Lakshmikantha, R. Pan, B. Prabhakar, and M. Seaman. Data center transport mechanisms: Congestion control theory and ieee standardization. In *46th Annual Allerton Conference on Communication, Control, and Computing*, 2008.

[4] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker. Packet caches on routers: the implications of universal redundant traffic elimination. In *SIGCOMM '08*, pages 219–230, New York, NY, USA, 2008. ACM.

[5] A. Anand, C. Muthukrishnan, A. Akella, and R. Ramjee. Redundancy in network traffic: findings and implications. In *SIGMETRICS '09*, pages 37–48. ACM, 2009.

[6] A. Anand, V. Sekar, and A. Akella. SmartRE: An architecture for coordinated network-wide redundancy elimination. In *SIGCOMM '09*, 2009.

[7] W. I. Caldwell, G. A. Coon, and L. M. Zoss. *Frequency Response for Process Control*. McGraw-Hill, 1959.

[8] J. L. Carter and M. N. Wegman. Universal classes of hash functions. In *STOC '77: Proceedings of the ninth annual ACM symposium on Theory of computing*, pages 106–112, New York, NY, USA, 1977. ACM.

[9] S. Chen and B. Bensaou. Can high-speed networks survive with droptail queues management? *Comput. Netw.*, 51(7):1763–1776, 2007.

[10] F. R. Dogar, A. Phanishayee, H. Pucha, O. Ruwase, and D. G. Andersen. Ditto: a system for opportunistic caching in multi-hop wireless networks. In *MobiCom*, 2008.

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

[12] K. Fall. A delay-tolerant network architecture for challenged internets. In *SIGCOMM '03*, pages 27–34, New York, NY, USA, 2003. ACM.

[13] J. Gray and P. Shenoy. Rules of thumb in data engineering. In *In Proc. 16th Internat. Conference on Data Engineering*, 2000.

[14] T. Grevers and J. Christner. *Application acceleration and wan optimization fundamentals.* Cisco Press, 2007.

[15] A. Gupta, A. Akella, S. Seshan, S. Shenker, and J. Wang. Understanding and exploiting network traffic redundancy. TR 1591, University of Wisconsin-Madison, Computer Science department, 2007.

[16] V. Jacobson, M. Mosko, D. Smetters, and J. J. Garcia-Luna-Aceves. Content-centric networking: Whitepaper describing future assurable global networks. Response to DARPA RFI SN07-12, 2007.

[17] P. Jokela, A. Zahemszky, C. Esteve, S. Arianfar, and P. Nikander. LIPSIN: Line speed publish/subscribe inter-networking. In *SIGCOMM*, 2009.

[18] D. Katabi, M. Handley, and C. Rohrs. Congestion control for high bandwidth-delay product networks. In *SIGCOMM '02*, pages 89–102, New York, NY, USA, 2002. ACM.

[19] T. Krovetz and P. Rogaway. Fast universal hashing with small keys and no preprocessing: The polyr construction. In *ICISC '00: Proceedings of the Third International Conference on Information Security and Cryptology*, pages 73–89, London, UK, 2001. Springer-Verlag.

[20] U. Legedza, D. Wetherall, and J. Guttag. Improving the performance of distributed applications using active networks. In *Infocom*, 1998.

[21] D. A. Patterson. Latency lags bandwith. *Commun. ACM*, 47(10):71–75, 2004.

[22] S. Paul, R. Yates, D. Raychaudhuri, and J. Kurose. The cache-and-forward network architecture for efficient mobile content delivery services in the future internet. In *Innovations in NGN: Future Network and Services. First ITU-T Kaleidoscope Academic Conference*, 2008.

[23] L. Roberts. Micro-flow management, Sept. 19 2006. US Patent App. 11/533,346.

[24] J. Santos and D. Wetherall. Increasing effective link bandwidth by suppressing replicated data. In *ATEC '98: Proceedings of the annual conference on USENIX Annual Technical Conference*, pages 18–18, Berkeley, CA, USA, 1998. USENIX Association.

[25] M. Särelä, T. Rinta-aho, and S. Tarkoma. RTFM: Publish/subscribe internetworking architecture. ICT Mobile Summit, 2008.

[26] N. T. Spring and D. Wetherall. A protocol-independent technique for eliminating redundant network traffic. In *SIGCOMM*, 2000.

[27] D. Trossen (ed.). Architecture definition, component descriptions, and requirements. Deliverable D2.3, PSIRP project, 2009.

[28] A. Vishwanath, V. Sivaraman, and M. Thottan. Perspectives on router buffer sizing: recent results and open problems. *SIGCOMM Comput. Commun. Rev.*, 39(2):34–39, 2009.

[29] L. wei H. Lehman, S. J. Garland, and D. L. Tennenhouse. Active reliable multicast. In *INFOCOM*, 1998.

[30] A. Wolman, M. Voelker, N. Sharma, N. Cardwell, A. Karlin, and H. M. Levy. On the scale and performance of cooperative web proxy caching. In *SOSP '99: Proceedings of the seventeenth ACM symposium on Operating systems principles*, pages 16–31, 1999.