On Content-Centric Router Design and Implications

Somaya Arianfar[1], Pekka Nikander[2] and Jörg Ott[1]
[1] Aalto University, [2] Ericsson Research, NomadicLab

ABSTRACT

In this paper, we investigate a sample line-speed content-centric router's design, its resources and its usage scenarios. We specifically take a closer look at one of the suggested functionalities for these routers, the content store. The design is targeted at pull-based environments, where content can be pulled from the network by any interested entity. We discuss the interaction between the pull-based protocols and the content-centric router. We also provide some basic feasibility metrics, discussing some applicability aspects for such routers.

1. INTRODUCTION

Research on future Internet architectures has pursued numerous directions. One suggestion is changing the architecture from a host-centric to a content-centric design [11], moving from the sender-controlled *send/receive* paradigm to a receiver-controlled model [17, 18], based on the *publish/subscribe* paradigm [10]. Recent research indicates that such "pub/sub" systems may offer improved robustness, usability, and security for both the network and the applications (e.g., [18, 19]). To achieve these improvements, the presented proposals suggest adding different kinds of functionality to the networking nodes, from serving the pending interests (subscriptions) to keeping a content store (cache) (e.g., [11]). However, the feasibility and the implications of adding these functions to the networking nodes have not been sufficiently discussed yet.

In this paper, we investigate the design of a *content-centric router*, and its usage scenarios. Specifically, we take a closer look at one of the suggested new functions for these nodes, the *content store*. The content store is, essentially, a storage area in the router that can be used as an explicit packet cache. It may even replace or use a large fraction of the memory used in today's routers for packet queuing; see Section 3.3 for the details. In our work, we assume a granularity of traditional packets for dealing with the content. We consider this choice as well-founded, as packets are a proven method for multiplexing and readily supported by networking equipment. This allows for incremental deployment of our content-centric routers.

Contrary to the current Internet, we look at content-centric routers as conjoint devices for both storing and forwarding information. We introduce a simple and efficient router design that can perform both of these operations at line-speed and that is able to serve, on its own, as a transient source of the content.

The design is targeted at pull-based environments, where any node interested interested in the content can *pull* it from the network [8, 11, 17]. This makes our work different from caching-enabled router proposals, e.g. [3, 6], which look at inter-packet redundancy elimination in push-based environments. In those designs, it is not only difficult to do caching at line-speed but it is also resource consuming. Their difficulty comes from the limiting factors of needing to encode and decode packets at different locations in the network and from requiring different levels of the co-ordination between the encoders, decoders, and other components of the system. Our design, on the other hand, relies on the content-centric networking principles and its pull-based logic. It introduces, network-wide, a set of content-centric routers that can work at line-speed without extra co-ordination.

We also provide an approximation on the required resources for our design. We believe that having content-centric routers as transient content sources can, as such, offer many functional and optimization benefits, expected from a shift toward content-centric networking. Our early evaluation results support this claim.

2. BACKGROUND

Content-centric network design fundamentally decouples the interests from the sources. Parties interested in a given piece of content do not need to know *where* to look for it. Hence, one can and has to assume that the content is identified, addressed, and matched independent of its location anywhere in the network.

Looking at the packet networks, they can only forward packets of up to some maximum packet size. Pieces of content larger than this limit require segmentation before they can be transmitted. A common design approach—which we also assume—is that each resulting segment is a uniquely identifiable piece of content in its own right, treated as an *independent* unit. Hence, for example, anything larger than 1280 bytes (the default minimum MTU size of IPv6) needs to be independently identifiable. As a result, fragmentation as such is not a big issue, as long as proper packet identi-

ACM ReArch 2010, November 30, 2010, Philadelphia, USA.

fiers are used. That is, each packet needs to be associated with a persistent identifier that is valid independent of any transport connections or other retrieval mechanisms.

In the *content-centric network (CCN)* design [11], content is explicitly addressable at all levels of the architecture. At the lowest level, packets—as the smallest pieces of content—have their own identities and attributes (e.g. for security) and can be addressed almost independently of anything else. The naming architecture follows a hierarchical structure. The matching between a piece of content and an expression of interest is based on these names. If the content name in an *interest* packet matches with the content-name (identifier) of a (cached) data-packet, it is considered a cache hit and the data-packet is forwarded to the requester.

PSIRP [17], as another content-centric proposal, uses a different approach to data naming. It follows an approach similar to DONA [13]'s flat labels, using a combination of cryptographic measures; it uses a cryptographic fingerprint computed over the piece of content or some other similar, algorithmically generated identifier. The content retrieval approach is then quite similar to CCN, at least at the abstract level.

With the aim of being independent of any specific naming proposals, in this paper we call the lowest level content identifiers as *packet identifiers (PIds)*. We assume a content-retrieval mechanism that follows the pub/sub model similar to CCN and PSIRP. That is, each content-centric router is able to forward, store, match, and send cached data packets, based on these PIds. We assume that the PId depends on the actual bits of the packet, allowing any receiver to distinguish between different versions of the whole content (/packet) it is requesting, avoiding the versioning problem typical to many caching systems.

In addition to identifying content, the system must also be able to identify destinations. That is, each router (and the actual data sources) need to be able to send data-packets back to the requester who has indicated interest in the content.[1] CCN solves this need by keeping pending interest tables in the routers, using this table to forward data back to the requesters. PSIRP allows for a set of different approaches, including ones where each packet carries a separate forwarding capability [12]; optionally, the PSIRP forwarding capabilities may even be content-dependent themselves [16]. Our design aims at allowing any suitable method; however, to keep the argumentation simple and focused on the content store, we assume that the router need not keep any per-request state. Instead, each request packet may carry a back-pointing forwarding identifier in itself.

3. CONTENT-CENTRIC ROUTERS

As a first step towards introducing a design for content-centric routers, we start by considering and organizing the router memory as an efficient cache *instead of* considering it merely as a packet buffer. We rely on the fact that once a packet has been written to the memory, it remains there until it is overwritten by another packet. There is no fundamental reason why a stored packet could not be addressed explicitly, allowing it to be re-used for purposes other than just queuing and pushing it out through an interface; in a way, this is similar to OS-level page sharing [14], but at a different level.

Although using caching in the routers is not a new idea on its own, in our design it is different as caching becomes a core part of the router functionality, in a way similar to what queuing is today. We make the caching an internal router operation rather than a network coordinated procedure. The resulting cache is called a content store, as it allows every single content-centric router to act as an independent source of cached content. In the following, we present our design in a content-centric environment, implicitly taking the router's speed, throughput, and other resources into account. In the next section, we then discuss the amount of resources required to support such a design.

3.1 Content store structure and operations

Our reference router model follows today's standard router memory hierarchy of CAM, SRAM, and DRAM. The content store (cache) contains two main components: a packet store and an index table to access the store; see Fig. 1. Packets are assigned to different locations in the packet store. The index table indexes the packets in the packet store.

The packet store is large and would be kept in the router's DRAM. The size of index table is smaller, and divided between DRAM and SRAM. The address for the incoming packets can be either pre-defined, using a part of the DRAM in a round-robin or random-replacement manner, or it can be calculated on the fly, based on available free space at the time of packet insertion. For now, we assume pre-computed fixed-sized cells, assigned by the PId indexing process.

3.1.1 Insertion and Deletion

As a starting point, we assume that all packets arriving at a router are indexed and cached; we relax that a little bit later. Each entry in the index table stores the PId, the packet's location in the DRAM, and some status information. These are divided between the SRAM and DRAM, aiming to minimize the amount of SRAM needed. However, there needs to be enough of information in the SRAM so that the router can determine, with high enough confidence, if it has stored a given packet or not, and when writing a new packet to the cache, to determine which packet to replace.

The design of the overall local caching system is depicted in Fig. 1. Each incoming packet is written to the available packet cell in the DRAM packet store, and its index information is updated at the index table.

Packet indexing may be implemented in different ways, trading off space and speed. Depending on the structure and the randomness of the PIds, the index address corresponding to a PId can be defined from a hash over PId, $H(PId)$, or from a some range of the PId bits. For the purpose of this paper, we consider index addresses that are directly drawn from the PId, *e.g.* bits{0...23}. The SRAM-stored part of an index entry needs to store a non-overallpoing range of the PId's bits, *e.g.* bits{24...51}. With this, the router can check from the SRAM if it has any packet with the bits {0..51} matching those of a given PId.

The easiest way to handle SRAM conflicts, i.e. packets where the address bits match but the stored bits do not, is to simply cache only one of the conflicting packets; see below. More elaborate schemes could use the SRAM as a hash table instead of using it as a simple array, with different

[1]The requests also need to be routed. CCN more-or-less relies on underlying IP network, with DNS, while PSIRP uses a DHT-like rendezvous. For this paper we simply assume that one of these many methods is used underneath.

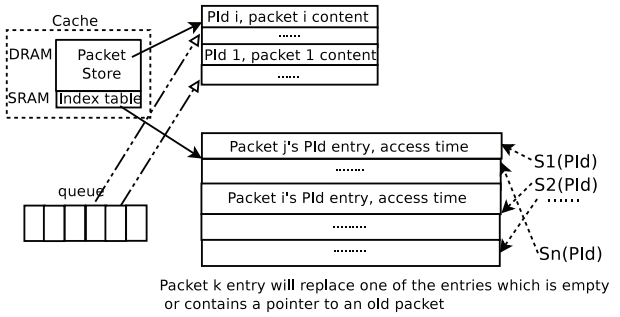

Figure 1: Structure of cache and queue in router's memory

bit ranges as addresses and/or buckets that can store more than one item.

To increase the probability of finding free space, one can employ different methods to calculate a number of candidate index addresses. For example, using parallel hash functions or choosing different ranges of the bits, $\{S_i\}$s, with evenly random distributed PIds, can result in multiple candidate addresses. In case some of the index entries point to occupied locations, there is still the possibility that at least one of them may point to a free location.

When considering a data packet for insertion, if there is an SRAM match, then the router may ignore the packet, since the probability of a false positive is around 2^{-52}. If there is no match, the router needs to store the packet and to select an index entry for this purpose. We could use, e.g., 8 bits in the SRAM to store status information, allowing the router select which index entry of the candidate ones to fetch from the DRAM and to update.

The extra bits in each SRAM entry can also help choosing which entry to replace. For instance, keeping the access history of the last few seconds in each entry can help to make cache replacement decisions similar to LRU. Of course, this LRU method is limited to the few candidate index positions, as doing the full LRU or similar methods would require long lookup times and would be too expensive for the router.

The proposed method of indexing and lookup, with storing part of an index entry in the SRAM and accessing the rest from the DRAM, can be compared to the BufferHash method, suggested in [5].

We follow a lazy deletion model, meaning that a stored packet is discarded only if a new packet arrives and needs to be inserted at that location. In this way, the new packet's index replaces the old packet's index in the index table and the new packet itself overwrites the existing packet in the packet store.

3.1.2 Lookup

Here, we propose a straightforward algorithm for packet lookup in the router's memory, aligned with our packet insertion method. Upon receiving a packet request, the router searches for the requested packet's PId in the index table. As explained above, the router first uses some bits from PId or $H(PId)$ as the address to an SRAM-located partial index entry; when using multiple sets, $\{S_i\}$s, the SRAM is checked for multiple potential partial index entries. After the potential index entry is located, it is fetched from the DRAM and compared against the rest of the bits in PId. If there is a match, the router has the wanted packet in the cache; otherwise there is a false positive.

3.2 Random autonomous caching

One of the main attributes that makes our design feasible is the ability of doing random autonomous caching in the content store. That is, the routers are able to cache data packets and process data requests randomly. In contrast to the other similar packet caching proposals, this provides the possibility of very *simple* load-sharing between routers. Since the design does not need encoding, decoding, and any other kinds of coordination, the decision over caching and processing a packet remains completely local to the router. Depending on how busy or interested a router is, a packet can either be both processed and forwarded, or only forwarded to the next router along the forwarding path.

3.3 Caching and queuing

In today's non-caching routers, assuming congestion, each received packet is buffered in DRAM and a pointer to it is enqueued in SRAM for scheduling. The same scheduling can still be done in our system, if needed. However, since we use hashes for random placement of the packets, there will be no FIFO tail competition similar to what exists between TCP flows today [9].

Depending on the details of the cache replacement algorithm, there may be a small probability of conflicts between caching and queuing requirements, if both the cache and the queue use the same memory. Consider a new packet that needs to be queued but happens to hash only to occupied locations. We then have two options. First, we can forcedly delete one of the already cached packets, freeing the location for the new packet. Second, 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 autonomous caching, 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 in a packet loss. This is no different from a queue overflow in current routers.

3.4 Packet identifiers and pull-based protocols

Compared to the limited caching proposals in the current Internet, the efficiency of our design partly relies on receivers knowing of the packet identifiers and the pull-based logic of the proposed content-centric transport protocols [8, 11, 18].

Most of the existing protocols are based on having a packet-naming entity somewhere upstream on the path. That is, there is a router upstream from the bottleneck link/path that continuously checks incoming packets, gives a name the new ones, and replaces cached packets with their names. With this information, a downstream router can then cache the newly named packets and replace any packet references (names) with the actual packets. Most of the time, these operations require the same amount of the memory and similar memory management policies at both ends of the bottleneck link. Being invisible to both upstream and downstream nodes, these methods do not affect the upward traffic or the server load. This appears to be the case, for instance, in parallel video streaming, when the server may become overloaded with sending the same data several times.

In our case, packet identification takes place at the end-hosts, leading to multiple advantages. First, it removes the need for an upward in-network entity to continuously name and encode the packets. Second, as a result of the first ad-

vantages, our method removes collaborative memory management requirements between the upward and downward routers on the bottleneck path. Third, it reduces the load on the upward path and toward the server. The third advantage comes from the combination of using pre-defined packet identifiers and using pull-based protocol where a request for an identified packet may never reach the server.

In pull-based content-centric protocols, the requester knows the packet PIds beforehand or can generate them locally. The requester then requests each of the packets (logically) separately; in practice, the requests can be generated in parallel, following a logic somewhat similar to the TCP.

As each content-centric router understands the request and response packets, it can easily cache responses (data packets) and reply to requests. If a request packet is answered from the cache, the request need not be forwarded. In this way, a router can independently reduce the load on the upward path, without any inter-router co-ordination effort. It is also expected to reduce the content retrieval time by the transport protocol.

4. RESOURCES

So far, we have discussed the *technical* feasibility of line-speed content-centric routers. While we do not claim we understand the overall complexity of applying our design to current routers, we provide some estimation of the individual router resources that such design would require. In this section, we give an approximation of the required resources for these kinds of routers, in terms of hardware and cost.

4.1 Processing throughput and memory latency

Considering processing, with a suitable ASIC or FPGA design, computing the index location, examining the resulting index entry, and writing the new index entry, takes at most a few clock cycles. The latency associated with writing or reading a packet to/from the DRAM takes the largest number of cycles, especially if slower and less power hungry DRAM is used.

Considering memory latency and 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. Given the typical DRAM random access time of around $50ns$, it is quite feasible to randomly access a memory bank once for every 2 or 7 packets for OC-192 and OC-768, respectively. Using several parallel banks and/or autonomous caching may be used to make sure that the DRAM latency does not become a bottleneck. It may also be possible to write multiple small packets at once, only keeping their SRAM indices separate. A more futuristic way of looking at the problem would be replacing DRAM with new chips, such as Reduced Latency DRAMs (RLDRAMs), that have current random access time of $15ns$ [1].

It is worth emphasizing the role of load sharing here compared to other router caching proposals. With the random autonomous caching and replying behavior, any operation that may appear heavy for the router can be easily passed to the next hop. For example, if a router is busy writing a packet to the memory, an arriving packet retrieval request that would require a read from the same memory bank can be passed unprocessed to the next hop. This kind of behavior keeps the overall system performance high. It also helps the individual routers to adjust their request serving rate based on their processing resources.

4.2 Storage

There is no definite limit on the amount of storage and, respectively, the time to cache packets. Routers today already have some amount of storage for buffering the packets. For example, the forthcoming 10G NetFPGA cards have three SRAM chips, each 72Mb, and four fast RLDRAM chips. Each RLDRAM chip is 512 Mb, together providing 250ms of full speed buffering capacity. The amount of SRAM in the router is limited mainly due to cost; however the amount of DRAM or RLDRAM is mostly kept relatively low, having capacity to buffer only 250ms worth of the packets, or even less; cf. [7].

From the caching point of view on the other hand, both the early (late 1990s) work on web caches [20] and the recent work on the time between the first and the last match in packet level caches [4], indicate that a few seconds of packet caching leads to major benefits. The results in [4] 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 to be 100 Gb, to keep all the traffic passing through an interface for 10 seconds, costing some $200–300, today.

Assuming a large packet size of 1500×8 bits, potentially typical for the kind of content-centric networking we are considering, indexing 100 Gb of data would need some 9 million index entries in SRAM. The size of the index table then would be dependent on the size of each entry.

In 10 seconds, some 9 million packets will pass through the router. We need to choose the SRAM-stored fraction of the PId so that it minimizes the false positive rate over these number of packets. In the case of our example of matching space of 52 bits between different packet PIds, with 9 million packets and 9 million SRAM entries of each consisting of 28 bits, the probability of a false positive is around 3.3%[2]. This rate seems to be acceptable for our content-centric routers. Using 52 bits for matching, with 24 bits used for addressing, means having index entries of 28 bits each. Adding 8 bits of status to that gives us 36 bit SRAM words.

Hence, the overall required SRAM for keeping the index table of 9 million 36 bit entries is around 324 Mb. Considering that commercial SRAM chips are commonly available 72 Mb (2M×36) configurations, with the assumed price of $125 per chip, the required 324Mb of SRAM for indexing large packets would cost some $500. Hence, our proposed content-centric caching memory would cost ≈$800 per line interface, for a $10s$ long full cache at 10 Gb/s. Given that multi-port line cards for commercial routers at 10 Gb/s have typical per-port price in the range of $1500–2500, our ballpark figure for a $10s$ long cache appears to be reasonable. The cost can be reduced by using methods such as random autonomous caching. The indiciations that memory prices drops faster than bandwidth [15] also help to argue that this cost would be more reasonable in the near future.

4.3 Energy consumption

While the price of the memory itself may not be a problem anymore, power consumption can become a limiting factor. In general, SRAM circuits consume significantly less power than typical DRAM circuits, allowing us to assume moderately fast SRAMs. From the DRAM side, our system can keep the total energy consumption reasonable by not re-

[2] $1 - ((1 - 2^{-28})^{9000000})) \approx 0.033$

quiring fast memory, using instead multiple banks of slower memory. 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, which is likely to be relatively small compared to the thermal load of the rest of the system. Assuming a high power cost of $0.20/kWh, the annual cost would be in the order of $20.

In our system, since caching is not done as a hard reliability component in the network, autonomous caching can be used reduce the average cost of individual routers while still keeping the overall efficiency high.

5. SIMULATIONS

To have some basic indications of the efficiency of our design, in terms of its interaction with our pull-based transport protocol and the effects of random autonomous caching, we have run some early ns-3 simulations. While our results are very early, they indicate that the design, when used in wide-scale with suitable pull-based protocols, could provide clearly improved traffic efficiency.

Our simulation settings consists of an ns-3 implementation of a native content-centric stack, using in-packet Bloom Filters [12] for forwarding and a new pull-based transport protocol [8] for content retrieval.

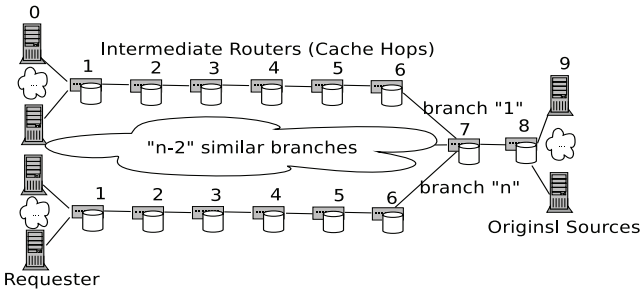

Figure 2: Example topology with requesters on the left, original sources on the right, and potential caches along the path.

In our simulations, we focused on the interactions between the content-centric router with random autonomous caching and our pull-based transport protocol. In each experiment, we had one or more transport flows, all trying to retrieve all the packets belonging to a bigger higher level content item, *e.g.*, a video clip. We used 8 different sets of the bits over the PIds for indexing. As a result, the overall write collision rate in our simulation was less than 0.3%. The basic scenario includes one of the branches shown in Fig. 2, with 8 routers in the path. Each intermediate router tries to cache the packets passing by it with the probability d.

In the first set of the experiments, we considered a high level content item that was first transfered once. A second request was then started within the caching lifetime of all of the original packets. As a result, requests for cached packets arrived at the first router on the path, and if not served, continued upward.

To study the effect of random autonomous caching, we varied the percentage of packets cached at each router, choosing the cached packets randomly with the probability d of $1, 1/2, 1/3, 1/4$, and $1/8$. The results, shown in Fig. 3, indicate that the overall mean efficiency can be quite high, even

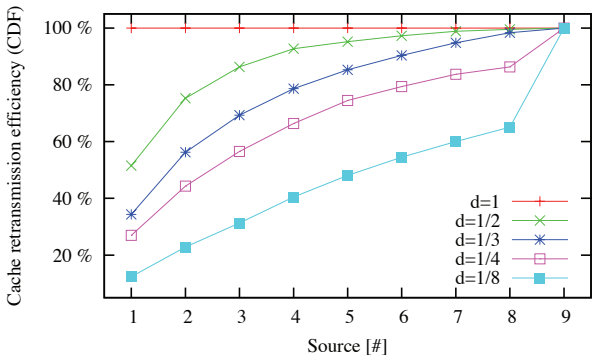

Figure 3: In-network cache efficiency for different caching probabilities in case of one flow in the network, with pre-cached contents.

with partial caching at each router. Obviously, when reducing the caching fraction, the number of routers needed for a given coverage increases; with full caching, the first cache essentially covers all the requests, while with 50% random caching, 4 caches achieve $\sim 95\%$ hit rate. For the relatively low caching fraction of 12.5%, the hit rate across 8 hops reached only $\sim 60\%$, indicating that some sort of inter-cache coordination might be useful at such low caching rates.

Our next simulation examined a more general content retrieval scenario, where all caches are initially empty, and they get filled in with real time traffic consisting of a number of parallel transfers. For this experiment, we started 8 distinct content retrieval flows, each requesting the same higher level piece of content simultaneously. 2 or 4 receivers from the different branches of Fig. 2 were chosen to pull the packets, in a rate controlled manner. The basic goal was to be as close as possible to real scenarios, where the data request rate varies at different times and caches happen to be created and replaced during the transfer. The intermediate routers' hop numbers indicate them as separate partial sources in the plots.

Fig. 4 illustrates the average amount of packets retrieved from different caches, with different caching rates. It can be seen that the requesters are able to retrieve a large proportion of the packets from the intermediate routers, reducing the traffic load on the upward path towards the source. The retrieval, though, is less efficient than in our previous experiment, because of the simultaneous filling of the caches.

Table 1 shows the time for each receiver to finish retrieving the whole content (Flow Completion Time: FCT). Comparing the results with the case with no caching (d=0), autonomous caching achieves a significant reduction of the FCT, even with low percentage of the caching in the routers. It is obvious that if there are larger number of simultaneous flows sharing the same bottleneck links toward the source, the model will achieve an even better level of the efficiency. It is important to note that this improvement is achieved solely based on the independent random caching in the content store and without any extra coordination effort.

6. CONCLUSION AND FUTURE WORK

In this paper, we have discussed a specific design with a content store for content-centric routers. This design is unique in the sense that it supports combined caching and

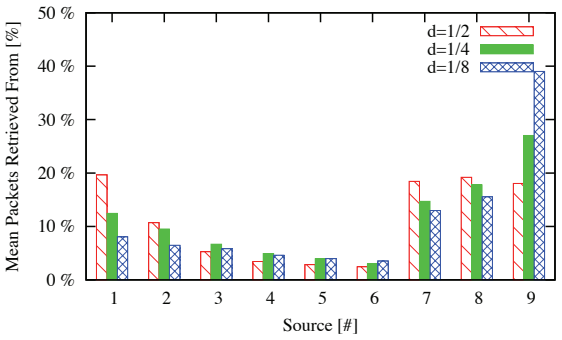

Figure 4: Mean percentage of packets received from each hop by 8 receivers. There are 8 simultaneous flows retrieving the same content.

d	Mean FCT(s)	Standard Deviation
0	1600.0	0.03
1/8	819.47	1.79
1/4	689.57	5.70
1/2	488.05	1.49

Table 1: Mean FCT of flows, requesting same content at same time.

queuing, random autonomous caching, and pull-based environments, without any need for caching-related co-ordination efforts between the routers. We have estimated the amount of resources required to support such a design. The design looks easily affordable compared to current routers, and has some potential to change the argument over dumb pipes. Our early evaluation results also indicate that the interactions between our content-centric routers and pull-based content retrieval protocol may result in a higher level of efficiency, compared to today's networks.

Our immediate future plan is to implement the system on the Stanford NetFPGA. Eventually, studying different combinations of push- and pull-based protocols, smarter ways of caching, and more realistic traffic patterns, are also part of our planned future work.

Acknowledgments

We would like to thank Aditya Akella for his valuable comments while preparing the final version of this paper. This work has been partially funded by the Finnish ICT SHOK Future Internet project.

7. REFERENCES

[1] Micron rldram. `http://www.micron.com/products/dram/rldram/`.

[2] Micron system power calculator. `http://www.micron.com/support/part\_info/powercalc.aspx`.

[3] Anand, A., Gupta, A., Akella, A., Seshan, S., and Shenker, S. Packet caches on routers: the implications of universal redundant traffic elimination. In *SIGCOMM '08* (2008).

[4] Anand, A., Muthukrishnan, C., Akella, A., and Ramjee, R. Redundancy in network traffic: findings and implications. In *SIGMETRICS* (2009).

[5] Anand, A., Muthukrishnan, C., Kappes, S., Akella, A., and Nath, S. Cheap and large cams for high performance data-intensive networked systems. In *NSDI'10: Proceedings of the 7th USENIX conference on Networked systems design and implementation* (Berkeley, CA, USA, 2010), USENIX Association, pp. 29–29.

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

[7] Appenzeller, G., Keslassy, I., and McKeown, N. Sizing router buffers. *SIGCOMM Comput. Commun. Rev. 34*, 4 (2004), 281–292.

[8] Arianfar, S., Eggert, L., Nikander, P., Ott, J., and Wong, W. Contug: A receiver-driven transport protocol for content-centric networks. Under submission, 2010.

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

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

[11] Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., and Braynard, R. L. Networking Named Content. In *Proc. ACM CoNEXT* (2009), pp. 1–12.

[12] Jokela, P., Zahemszky, A., Esteve Rothenberg, C., Arianfar, S., and Nikander, P. LIPSIN: Line Speed Publish/Subscribe Inter-Networking. In *Proc. ACM SIGCOMM* (2009), pp. 195–206.

[13] Koponen, T., Chawla, M., Chun, B.-G., Ermolinskiy, A., Kim, K. H., Shenker, S., and Stoica, I. A data-oriented (and beyond) network architecture. In *SIGCOMM '07* (2007).

[14] Milos, G., Murray, D. G., Hand, S., and Fetterman, M. Satori: Enlightened page sharing. In *Usenix* (2009).

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

[16] Rothenberg, C., Jokela, P., Nikander, P., Sarela, M., and Ylitalo, J. Self-Routing Denial-of-Service Resistant Capabilities using In-Packet Bloom Filters. In *Proc. EC2ND* (2009).

[17] Trossen (ed.), D. Architecture Definition, Component Descriptions, and Requirements. Deliverable D2.3, PSIRP Project, 2009.

[18] Trossen (ed.), D. Update on the Architecture and Report on Security Analysis. Deliverable D2.4, PSIRP Project, 2009.

[19] Trossen (ed.), D. Final Updated Architecture. Deliverable D2.5, PSIRP Project, 2010.

[20] Wolman, A., Voelker, M., Sharma, N., Cardwell, N., Karlin, A., and Levy, H. M. On the scale and performance of cooperative web proxy caching. In *SOSP '99* (1999), pp. 16–31.