Rule-based Stream Reasoning for Intelligent Administration of Content-Centric Networks

Harald Beck[1], Bruno Bierbaumer[2], Minh Dao-Tran[1], Thomas Eiter[1], Hermann Hellwagner[2], and Konstantin Schekotihin[2]

[1] TU Wien, Vienna, Austria {beck,dao,eiter}@kr.tuwien.ac.at
[2] Alpen-Adria-Universität Klagenfurt, Austria
bruno@itec.aau.at, firstname.lastname@aau.at

Abstract. Content-Centric Networking (CCN) research addresses the mismatch between the modern usage of the Internet and its outdated architecture. Importantly, CCN routers use various *caching strategies* to locally cache content frequently requested by end users. However, it is unclear which content shall be stored and when it should be replaced. In this work, we employ novel techniques towards intelligent administration of CCN routers. Our approach allows for autonomous switching between existing strategies in response to changing content request patterns using rule-based stream reasoning framework LARS which extends Answer Set Programming for streams. The obtained possibility for flexible router configuration at runtime allows for faster experimentation and may result in significant performance gains, as shown in our evaluation.

1 Introduction

Various future Internet research efforts are being pursued for efficient multimedia distribution, among them *Content-Centric Networking* (CCN) [14]. The operation of a CCN network relies on two packet types, *Interest* and *Data*. Clients issue *Interest* packets containing the *content name* they want to retrieve. CCN routers forward the *Interest* packets until they reach a content provider, which answers with a *Data* packet. The latter travels back to the content consumer following the way of *Interest* packets. In addition, the CCN routers have the possibility to cache *Data* packets in their *Content Stores*. Thus, the *Interest* packets of another consumer can be directly satisfied out of a *Content Store*. These caches make it possible to satisfy popular content requests directly out of caches and reduce the network load [14].

A *caching strategy* defines which content is stored and for how long before being replaced. There is a rich literature of strategies for CCN [24, 21]. Examples of most popular strategies include: (a) *Least Recently Used* (LRU), which orders items in cache by access time stamps and replaces the oldest item; (b) *First-In-First-Out* (FIFO) implementing a queue; (c) *Least Frequently Used* (LFU) which orders items by access frequency and replaces the least accessed item; or (d) *Random* that replaces a random item in the cache. However, selection of an appropriate strategy is complicated.

Example 1 Consider a situation in which some music clips go viral, i.e., get very popular over a short period of time. In this case, network administrators may manually configure the routers to cache highly popular content for some time period, and to switch

back to the usual caching strategy when the consumer interests get more evenly distributed. However, as this period of time is hard to predict, it would be desirable that routers autonomously switch their caching strategy to ensure high quality of service. ■

Evaluations, like [4, 24], show that no "silver bullet" strategy is superior in all tested scenarios, since for every strategy there are conditions in which it works best. There conditions can often be characterized by parameters of a consumer interests distribution. Usually, the content popularity is described in the literature with a *Zipf* distribution [18]: $P(X = i) = \left(i^{\alpha} \sum_{j=1}^{C} 1/j^{\alpha}\right)^{-1}$, where C is a number of items in the content catalog, α is a value of the exponent characterizing the distribution and i is a rank of an item in the catalog. The variation of the exponent α allows to characterize different popularity models for consumers interests: (i) if α is high, the popular content is limited to a small number of items; (ii) if α is low, every content is almost equally popular.

As real CCNs are not deployed yet, there is currently no real-world experience to rely on, and developing selection methods for caching strategies is not well supported. Motivated by all this, we consider a router architecture that allows for dynamic switching of caching strategies in reaction to the current network traffic, based on *stream reasoning*, i.e., reasoning over recent snapshots of data streams.

Contributions. (i) We present an *Intelligent Caching Agent* (ICA) for the administration of CCN routers using stream reasoning, which allows for the first implementation of a local and dynamic caching strategy selection. (ii) To simulate various CCN application scenarios, router architectures and rule-based administration policies, we propose an extension of the well-known CCN simulator ndnSIM [15]. (iii) The evaluation results of our methods on two sample scenarios (as in Example 1) indicate a clear performance gain when basic caching strategies are dynamically switched by routers in reaction to the observed stream of requested data packets.

In summary, we provide a feasibility study for using logic-based stream reasoning techniques to guide selection of caching strategies in CCNs. Moreover, we also provide a detailed showcase of analytical, declarative stream reasoning tools for intelligent administration problems; to the best of our knowledge, no similar work exists to date.

2 Stream Reasoning

Router administration requires evaluation of streaming data. To the best of our knowledge, declarative *stream reasoning* [6] methods [12, 11, 23, 16, 2] have not been used.

Example 2 (con't) Consider the following rules to select a caching strategy. If in the last 30 seconds there was always a high $\hat{\alpha}$ value (some content is very popular), use LFU, and for a medium value, take LRU. Furthermore, use FIFO if the value is low but once in the last 20 seconds 50% was real-time content. Otherwise, use Random. ■

Example 2 illustrates that a fully declarative, rule-based language would assist the readability of a router's module that controls (potentially far more complex) decisions. We employ the rule-based LARS [2] which can be seen as extension of Answer Set Programming (ASP) [3, 10] for streams. In particular, it provides *window* operators to limit limit reasoning to so-called *snapshots* as in CQL [1]. We give a high-level intuition.

$r_1:\ @_T high \leftarrow \boxplus^{30} @_T \hat{\alpha}(V),\ V \geq 1.8.$
$r_2:\ @_T mid \leftarrow \boxplus^{30} @_T \hat{\alpha}(V),\ 1.2 \leq V < 1.8.$
$r_3:\ @_T low \leftarrow \boxplus^{30} @_T \hat{\alpha}(V),\ V < 1.2.$
$r_4:\ use(lfu) \leftarrow \boxplus^{30} \Box high.$
$r_5:\ use(lru) \leftarrow \boxplus^{30} \Box mid.$
$r_6:\ use(fifo) \leftarrow \boxplus^{30} \Box low,\ \boxplus^{20} \Diamond rtm50.$
$r_7:\ done \leftarrow use(lfu) \vee use(lru) \vee use(fifo).$
$r_8:\ use(random) \leftarrow \text{not } done.$

Fig. 1: Program P deciding which caching strategy to use

LARS. A LARS program is a set of rules of form $\alpha \leftarrow \beta_1, ..., \beta_j, \text{not } \beta_{j+1}, ..., \text{not } \beta_n$, $(n \geq 0)$ where $\alpha, \beta_1, \ldots, \beta_n$ are formulas and not denotes *negation-as-failure*. Let a be an atom and $t \in \mathbb{N}$. Then, the set $\mathcal{F}$ of LARS *formulas* is defined by the grammar $\varphi ::= a \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \rightarrow \varphi \mid \Diamond\varphi \mid \Box\varphi \mid @_t\varphi \mid \boxplus^w\varphi$. It uses the following:

- The *window operator* $\boxplus^w$ limits evaluation of a formula φ to the substream returned by a function w, which takes a stream and a time point. We only use a special window operator $\boxplus^k$ which returns the snapshot of the last k seconds.
- The *temporal quantifiers* $\Diamond$ and $\Box$ are used to query whether a formula φ holds at *some* time point in a selected window, or at *all* time points.
- The $@$*-operator* allows a jump in time, i.e., $@_t\varphi$ evaluates φ at time t.

Example 3 Fig. 1 formalizes the rules of Example 2 in LARS, where atom $\hat{\alpha}(V)$ is used to retrieve from the router an estimation of the α value V. Similarly, $rtm50$ is true if at least 50% of the content forwarded by the router was real-time. Rule (r_1) says the following. If in the last 30 seconds ($\boxplus^{30}$), at a specific (variable) time T ($@_T$) we had atom $\hat{\alpha}(V)$ for some value $V \geq 1.8$, then $high$ is true at T. Then, rule (r_4) states that, if $high$ is true at *all* ($\Box$) of the last 30 seconds, then ($\leftarrow$) we shall use lfu. If $use(X)$ cannot by derived for any $X \in \{lfu, lru, fifo\}$ by rules $(r_4) - (r_6)$, the disjunction in (r_7) fails, thus $done$ will not be derived, and due to (r_8) we will then use $random$. ∎

3 System Description

As shown in Fig. 2, ICA extends the architecture of a common CCN router with a decision unit, which consists of three main components: (1) a database (DB) storing snapshots of parameters observed by the controller, (2) a knowledge base (KB) containing the ICA logic and (3) a reasoner that decides about configuration of the controller given the KB and a series of events in the DB. This architecture was implemented in ndnSim [15] and used in the evaluation as presented in Section 4.

The components (2) and (3) are based on the LARS framework, which we implemented using DLVHEX 2.5 [9] as language of this system, i.e., higher-order logic programs with external atoms. We define an external atom $\&w[S, E, F](T, V)$ representing the described time-based LARS window operator. The terms $S, E \in \mathbb{N}$ define the time interval of the window and F is a string comprising a function name. Our DLVHEX plug-in evaluates the function over events registered in the database within the given time interval and returns its results as a set of tuples $\{(t_1, v_1), \ldots, (t_k, v_k)\}$,

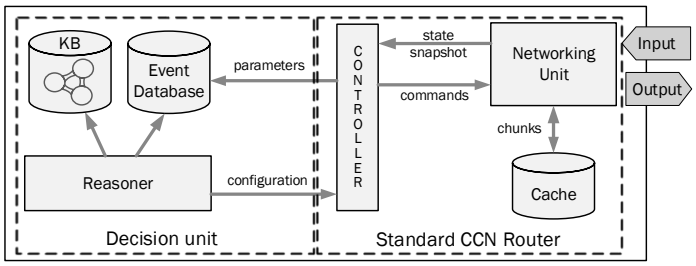


Fig. 2: Architecture of an Intelligent Caching Agent (ICA)

```
1 intv1(S,E) :- &getSolverTime[](E), S=E-30.
2 intv2(S,E) :- &getSolverTime[](E), S=E-20.

3 val(high,S,E,T):- &w[S,E,alpha](T,V), V>=18, intv1(S,E).
4 val(mid,S,E,T) :- &w[S,E,alpha](T,V), 12<=V, V<18, intv1(S,E).
5 val(low,S,E,T)  :- &w[S,E,alpha](T,V), V<12, intv1(S,E).
6 val(rtm50,S,E,T):- &w[S,E,rtc](T,V), V>50, intv2(S,E).

7 some(ID,S,E)   :- val(ID,S,E,_).
8 always(ID,S,E) :- val(ID,S,E,_), val(ID,S,E,T):T=S..E.

9 use(lfu) :- always(high,S,E), intv1(S,E).
10 use(lru) :- always(mid,S,E),  intv1(S,E).
11 use(fifo):- always(low,S1,E1), intv1(S1,E1), some(rtm50,S2,E2), intv2(S2,E2).

12 done :- use(X), X!=random.
13 use(random) :- not done.
```

Listing 1.1: DLVHEX encoding for ICA

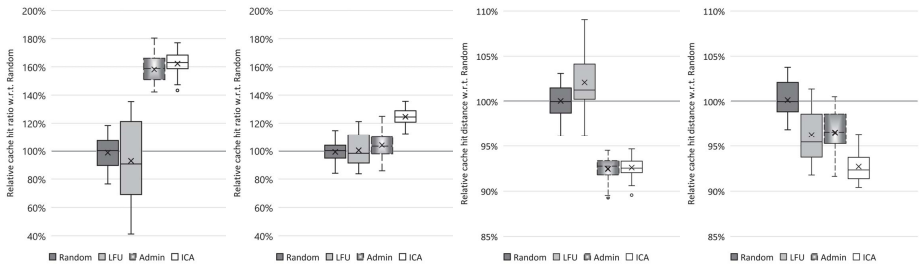
where t_i and v_i indicate the time point and the value of a function, respectively. E.g. for $F =$ alpha the estimated values $\hat{\alpha}$ of the parameter α of the Zipf distribution will be returned. To define rules that respect only recent events, we use an external atom $\&getSolverTime[](E)$ which has no inputs. It outputs the current system time E. The DLVHEX encoding for ICA is presented in Listing 1.1, which corresponds to the LARS encoding presented in Fig. 1 and could be in principle automatically generated from it.

4 Evaluation

We selected the *Abilene* topology [20]. For every simulation run (see below), we connected 1000 consumers and all content providers randomly to one of the 11 routers.

Scenarios. As popularity change scenarios, we used (i) *LHL* that starts with $\alpha = 0.4$ (low), then changes to 2.5 (high), and then back to low; (ii) *HLH* is dual. The values are from $[17, 21]$. Each simulation is 1800 seconds, α changes at 600 and 1200. Each consumer starts downloading a video at a random time point in each interval.

Caching Strategies. To measure the potential effect of switching strategies, we compare against the static ones *Random* and and LFU [21]. Dynamic strategy *Admin* is hypothetical, where all routers change their caching strategy exactly at phase changes L



(a) LHL cache hit ratio (b) HLH cache hit ratio (c) LHL cache hit dist. (d) HLH cache hit dist.

Fig. 3: Aggregated evaluation results over 30 runs for each caching strategy

to H and H to L; in L they use Random, in H they use LFU. Finally, *Intelligent Caching Agent (ICA)* dynamically selects for each router a strategy due to locally observed data.

Simulation System Parameters. Following [17, 14, 19], we use 1000 users $\times$ 50 videos, in 1000 chunks of 10KB. Routers store 0.1, 0.5, 1, 4 or 10% of all chunks.

Performance Metrics. The *cache hit ratio* should be high; it is the number of hits an *Interest* packet is satisfied by a router's content store per total number of requests. The *cache hit distance* should be low; it is the average number of *hops* for a *Data* packet from request to a router that returns it, i.e., the number of routers travelled between the router answering a request and the consumer that had issued it. See [24] for details.

Results. We determined 1% of chunks to be a reasonable storage size. We observed that the reaction to changing content access for ICA was close to the ideal preconfiguration of Admin. Interestingly, up to 5 routers used the Random strategy in the H phase under the ICA strategy. Here, the advantage of dynamic and *local* switching kicked in.

Fig. 3 shows performance comparisons of caching strategies LFU, Admin and ICA in relation to Random (100%), where plots show aggregated results over 30 individual runs. Fig. 3a/3b depict cache hit ratios for LHL/HLH; Fig. 3c/3d show cache hit distances. In summary, dynamic switching is advantageous in all settings. ICA is at least as good as Admin for LHL scenarios, and proves to be the best strategy for HLH due to the advantage of choosing strategies locally for each router. Notably, both dynamic strategies lead to a decreased cache hit distance relative to the Random strategy.

5 Conclusion

In our paper we focused on a principled approach of automated decision making by means of high-level reasoning on stream data. This allowed us to design a purely declarative control unit for automated administration of CCN routers. A comprehensive feasibility study shows how reasoning techniques can be used for dynamic switching of caching strategies in reaction to changing user behavior may give significant savings due to performance gains. These observations clearly motivate the advancement of stream reasoning research, especially on the practical side. In particular, stream processing engines are in need that have an expressive power similar to LARS.

References

1. Arasu, A., Babu, S., Widom, J.: The CQL continuous query language: semantic foundations and query execution. VLDB J. 15(2), 121–142 (2006)
2. Beck, H., Dao-Tran, M., Eiter, T., Fink, M.: LARS: A Logic-based Framework for Analyzing Reasoning over Streams. In: AAAI. (2015)
3. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Communications of the ACM 54(12), 92–103 (2011)
4. Cha, M., Kwak, H., Rodriguez, P., Ahn, Y., Moon, S.B.: Analyzing the video popularity characteristics of large-scale user generated content systems. IEEE/ACM Trans. Netw. 17(5), 1357–1370 (2009)
5. Cisco Visual Networking Index: Forecast and Methodology, 2014-2019. White Paper (2016)
6. Della Valle, E., Ceri, S., van Harmelen, F., Fensel, D.: It's a Streaming World! Reasoning upon Rapidly Changing Information. IEEE Intelligent Systems 24, 83–89 (2009)
7. Do, T.M., Loke, S.W., Liu, F.: Answer Set Programming for Stream Reasoning. In: Adv. Artif. Intell. pp. 104–109 (2011)
8. Eiter, T., Fink, M., Krennwallner, T., Redl, C.: Domain Expansion for ASP-Programs with External Sources. Artif. Intell. 233, 84–121 (2014)
9. Eiter, T., Mehuljic, M., Redl, C., Schüller, P.: User guide: dlvhex 2.x. Tech. Rep. INFSYS RR-1843-15-05, TU Vienna (2015)
10. Faber, W., Leone, N., Pfeifer, G.: Recursive aggregates in disjunctive logic programs: Semantics and complexity. In: JELIA. pp. 200–212 (2004)
11. Gebser, M., Grote, T., Kaminski, R., Obermeier, P., Sabuncu, O., Schaub, T.: Stream Reasoning with Answer Set Programming. Preliminary Report. In: KR. pp. 613–617 (2012)
12. Gebser, M., Kaminski, R., Kaufmann, B., Ostrowski, M., Schaub, T., Thiele, S.: Engineering an incremental ASP solver. In: ICLP. pp. 190–205 (2008)
13. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New Gener. Comput. 9(3-4), 365–386 (1991)
14. Jacobson, V., Smetters, D.K., Thornton, J.D., Plass, M.F., Briggs, N.H., Braynard, R.: Networking named content. In: CoNEXT. pp. 1–12. (2009)
15. Mastorakis, S., Afanasyev, A., Moiseenko, I., Zhang, L.: ndnSIM 2.0: A new version of the NDN simulator for NS-3. Technical Report NDN-0028, NDN (2015)
16. Mileo, A., Abdelrahman, A., Policarpio, S., Hauswirth, M.: Streamrule: A nonmonotonic stream reasoning system for the semantic web. In: RR. pp. 247–252. (2013)
17. Rossi, D., Rossini, G.: Caching performance of content centric networks under multi-path routing (and more). Relatório técnico, Telecom ParisTech (2011)
18. Rossi, D., Rossini, G.: On sizing CCN content stores by exploiting topological information. In: IEEE INFOCOM. pp. 280–285 (2012)
19. Rossini, G., Rossi, D., Garetto, M., Leonardi, E.: Multi-terabyte and multi-gbps information centric routers. In: IEEE INFOCOM. pp. 181–189. (2014)
20. Spring, N.T., Mahajan, R., Wetherall, D., Anderson, T.E.: Measuring ISP topologies with rocketfuel. IEEE/ACM Trans. Netw. 12(1), 2–16 (2004)
21. Tarnoi, S., Suksomboon, K., Kumwilaisak, W., Ji, Y.: Performance of probabilistic caching and cache replacement policies for content-centric networks. In: IEEE LCN. pp. 99–106. (2014)
22. Yu, H., Zheng, D., Zhao, B.Y., Zheng, W.: Understanding user behavior in large-scale video-on-demand systems. In: EuroSys. pp. 333–344. (2006)
23. Zaniolo, C.: Logical foundations of continuous query languages for data streams. In: Datalog. pp. 177–189 (2012)
24. Zhang, M., Luo, H., Zhang, H.: A Survey of Caching Mechanisms in Information-Centric Networking. IEEE Communications Surveys and Tutorials 17(3), 1473–1499 (2015)