Efficiently routing bandwidth demands with incremental stream computation

Nicola Rustignoli
Open Systems AG, Switzerland
rnicola@student.ethz.ch

Desislava Dimitrova*, Sebastian Wicki, Timothy Roscoe
Systems Group, ETH Zurich, Switzerland
(dimitrova,sebastian.wicki,roscoe)@inf.ethz.ch

1 INTRODUCTION

In routing, the ability to quickly react to network changes is essential for ensuring network availability and efficient operation. Adapting to link utilization can offer benefits for traffic management in both wide-area networks (by more flexible allocation of expensive link resource) and datacenter networks (by giving preferential treatment to group of flows). Many studies address the problem of bandwidth-constraint routing from an algorithmic perspective. We are interested in providing a faster, more efficient execution platform for those algorithms. Previously, we demonstrated that executing routing as an incremental stream computation can quickly recover from failures [2]. Here, we extend the proposal to support bandwidth-constraint routing.

We **contribute** (1) an approach to quickly execute a range of bandwidth-constraint routing algorithms, (2) a prototype implementation inside a stream processing system, and (3) an initial set of evaluations for both datacenter and WAN scenarios.

2 EFFICIENT ROUTING EXECUTION

Topology model. The network topology is represented as a property graph $G = (V, E)$ where V is a set of nodes, and E is a set of edges. Edges are identified by their end points (e.g., s_1, s_2) and have associated properties: utilization, *cost* and a $delta$ value $\delta \in \{-1, +1\}$. **Link cost** is defined by a cost function and we implemented three versions: widest-path routing (a min/max cost function) and two versions of shortest distance routing (an additive cost function) [3] using link utilization and free bandwidth, respectively. We present results for the latter. **Link update** is modelled as a sequence of edge removal (s_1, s_2, $cost$,$\delta = -1$) and edge addition with updated cost (s_1, s_2, $cost_new$, $\delta = 1$).

Incoming **flow requests** are defined on G and represented as $flow ::= (flowID, S, T, BW)$, where $flowID$ is an ID that uniquely identifies the flow, S and T are the flow's origin and target nodes respectively, and BW is the requested bandwidth.

Execution model. We propose a routing execution model which is *proactive* and handles updates incrementally. Our system ingests streams of link updates and flow reservations, and internally maintains a set of forwarding rules R which tracks the path with most free bandwidth between any pair of nodes.

Rules are initialized on G, after which we process a bandwidth reservation request in three stages. First, from the precomputed rules, we materialize a candidate path with sub-millisecond latency. Second, admission control checks if the path can offer the requested bandwidth, performs necessary flow bookkeeping and constructs updates for the affected links. Finally, the stream of link updates is fed to an incremental routing computation, which updates the set of forwarding rules. As an optimization we allow network updates to be **batched** until a certain link utilization threshold is reached, i.e., lazy updates. This significantly reduces the number of forwarding rule recomputations, at the cost of less balanced link utilization.

Implementation. Each stage in the model is implemented as a streaming operator in Timely Dataflow. The platform is a natural fit to routing: it has *event-driven* programming model and native support for arbitrary *iterative computations*. Furthermore, in our experience, Timely's Rust implementation outperforms other platforms such as Flink [1] and Spark Streaming [4].

3 EVALUATION

As a proof-of-concept experiment we present throughput evaluations for a datacenter network and a wide area network.

Datacenter. We use FatTree topology built with 48-port switches; there are 2880 switches in total and links have total capacity of $10 Gbps$. From the pool of access switches we randomly select 10k pairs of source destinations, corresponding to flows. Flow rates and inter-arrival times follow distributions for cache followers in the Facebook trace [5]. We use 32 workers for rerouting. As we increase the lazy updates threshold from 0K (no batching) up to $1 Gbps$ (10% of link capacity), throughput steadily increases up to $2k$ requests/sec. Thresholds lower than the average flow rate ($5 Mbps$) do not gain from batching, explaining the plat performance.

WAN. We chose five topologies from the Internet Topology Zoo (ITZ) to cover different size graphs: small (Gridnet), medium (Abeline, AT&T) and large (Cogent,Colt). Link capacities are as in ITZ. We then submit thousand traffic requests compliant with randomised traffic matrix generation. For small graphs single-worker computation suffice, for bigger 8 workers are used. Throughput already starts at 200 requests/sec, for smaller topologies even above 500 reservations/sec. This is an order of magnitude higher than the datacenter case. The reason - WAN topologies have an order of magnitude less nodes than FatTree (200 nodes at most). High thresholds rarely trigger rerouting, leading to the same upper bound on throughput in both WAN and datacenter.

REFERENCES

[1] Chothia, Z., Liagouris, J., Dimitrova, D., and Roscoe, T. Online reconstruction of structural information from datacenter logs. In *Proceedings of European Conference on Computer Systems* (2017), EuroSys '17, ACM, pp. 344–358.

[2] Dimitrova, D. C., Liagouris, J., Hoffmann, M., Kalavri, V., Wicki, S., and Roscoe, T. Quick incremental routing logic for dynamic network graphs. In *SIGCOMM Posters and Demos '17* (2017), ACM, pp. 9–11.

[3] Ma, Q., and Steenkiste, P. On path selection for traffic with bandwidth guarantees. In *International Conference on Network Protocols* (1997), pp. 191–202.

[4] McSherry, F., Isard, M., and Murray, D. G. Scalability! but at what COST? In *15th Workshop on Hot Topics in Operating Systems (HotOS XV)* (2015), USENIX.

[5] Roy, A., Zeng, H., Bagga, J., Porter, G., and Snoeren, A. Inside the social network's (datacenter) network. *SIGCOMM Comp. Comm. Rev. 45*, 4 (2015), 123–137.

Efficiently routing bandwidth demands with incremental stream computation

Nicola Rustignoli, Desislava Dimitrova, Sebastian Wicki, Timothy Roscoe

NETWORK UTILIZATION MATTERS

Workloads evolve and diversity.
Routing should map demands to resource.
Quickly.

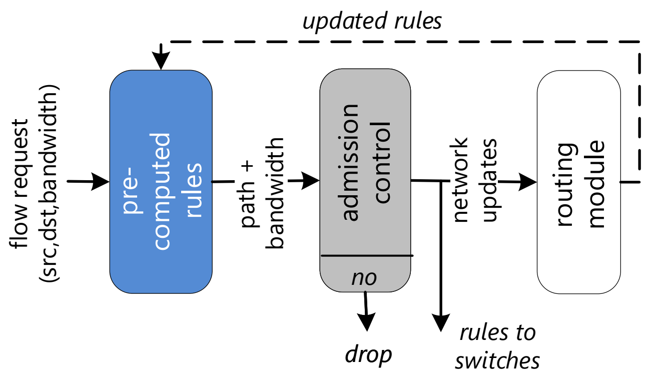
OUR PROPOSAL

Efficient, *proactive* routing of traffic demands with *incremental* computation and *batched* request processing.

BANDWIDTH-CONSTRAINT ROUTING

PROACTIVE COMPUTATION

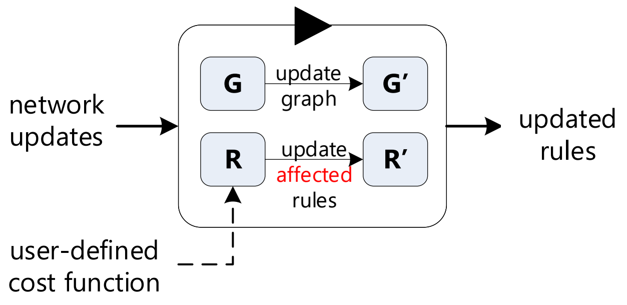
Forwarding rules are precomputed for fast query.



Flow rule	src	dst	nextHop	cost	Δ=1
Network update	src	dst	cost_old	Δ=-1	
	src	dst	cost_new	Δ=1	

INCREMENTAL RE-ROUTING

Upon network update recompute ONLY affected rules.



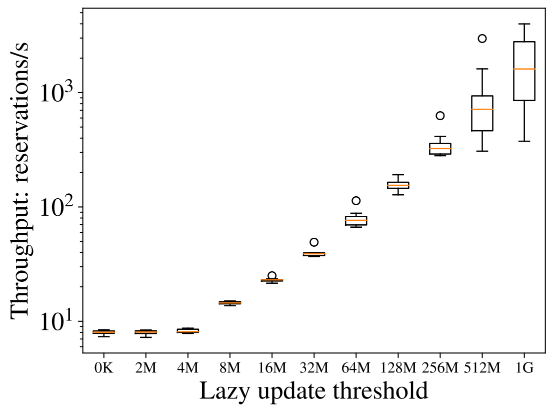
LAZY UPDATES OPTIMIZATION

1. Store network updates until aggregated bandwidth reaches an update threshold.

2. Push network updates in a single batch.

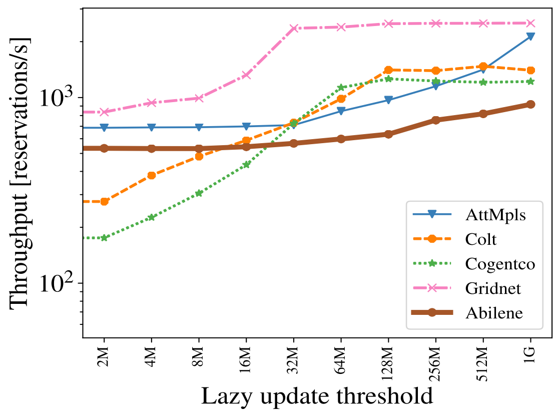
EVALUATIONS

DATACENTER CASE



Up to **2k flow requests per second** in a 48-ary FatTree with 32 workers. *Facebook workload.*

WIDE AREA NETWORK CASE



Constant high gain for small topologies and increasing for big ones. *Randomised traffic matrices.*

SUPPORTED ALGORITHMS

- *Cumulative cost*
  - hop count
  - shortest distance link utilization
  - shortest distance free bandwidth
- *Max/Min cost*
  - shortest-widest path

dimitrova@inf.ethz.ch