# Poster: Raptor: Rapid prototyping of distributed stream processing applications at scale

Md. Monzurul Amin Ifath
Dalhousie University
monzurul.amin@dal.ca

Miguel Neves
Dalhousie University
mg478789@dal.ca

Israat Haque
Dalhousie University
israat@dal.ca

## ABSTRACT

Stream processing applications are becoming increasingly important in areas such as IoT, video analytics and social media. As a result, developers and operators must meet stringent time-to-market and scale requirements before bringing them to production. Unfortunately, testing a networked stream processing system is currently a cumbersome process that usually requires an expensive testbed and deep expertise on both networking and distributed systems. In this poster, we present Raptor, a tool for the fast prototyping of large-scale networked stream processing applications. Raptor builds on Mininet and Apache Kafka, two widely adopted platforms, to enable stakeholders to easily test their solutions under various operational conditions. Through a reasonably large setup (20 nodes) running on a single server, we show how unbalanced Kafka's leader selection algorithm can be and its implications on the overall system's throughput. We envision this work can help paving the way for more reproducible research in the stream processing domain, currently a first-class network application.

## CCS CONCEPTS

• **Networks** → **Network measurement**; **Network simulations**.

## KEYWORDS

Stream processing, Network applications, Reproducibility

**ACM Reference Format:**
Md. Monzurul Amin Ifath, Miguel Neves, and Israat Haque. 2021. Poster: Raptor: Rapid prototyping of distributed stream processing applications at scale. In *The 17th International Conference on emerging Networking EXperiments and Technologies (CoNEXT '21), December 7–10, 2021, Virtual Event, Germany.* ACM, New York, NY, USA, 2 pages. https://doi.org/10.1145/3485983.3493354

## 1 INTRODUCTION

In recent years, we have seen a surge on the number of applications (e.g., social media [2], finance [3], healthcare [4]) that rely on the processing of data streams. Indeed, more than 80% of the Fortune 100 companies currently use some stream processing platform, e.g., Apache Flink, Kafka Streams [1]. Unlike traditional batch processing where data is collected over time and then analyzed, stream processing applications continuously query and analyze data, which enables faster response. Despite the great success of the stream processing paradigm, testing a networked stream processing application (particularly at scale) is still a cumbersome and often expensive process. On one hand, developers and operators must cope with all challenges associated to deploying a networked system (e.g., routing, addressing, monitoring). On the other hand, they also need to rely on either costly testbeds (usually a cluster of servers organized into some pre-defined, i.e., fixed topology) or complex cloud-based setups for running their experiments. Although there exist a few virtual testbeds available to the research community for free, e.g., GENI, CloudLab and Chameleon, they normally require advanced expertise from users to create a timely application setup. Ultimately, these challenges delay new advancements in the area and sometimes even forbid interested parties to participate on it.

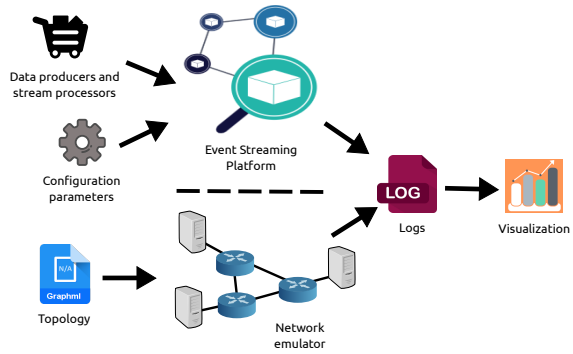


**Figure 1: Raptor workflow.**

This work presents Raptor, a tool designed to lower the cost (i.e., time and money) of prototyping and experimenting with large-scale networked stream processing applications. Raptor combines the flexibility and realism of a network emulator (Mininet [5]) with the generality of a widespread event streaming platform (Apache Kafka) to provide developers an easy way to test their solutions under multiple operational conditions (e.g., different network loads and topologies, replication mechanisms, failure models). The tool can run reasonably large setups (beyond 20 nodes) on a commodity server and offers a great set of detailed measurements, including bandwidth and latency reports as well as event logs. We hope Raptor can also facilitate the reproducibility of research results in the stream processing domain.

## 2 RAPTOR OVERVIEW

This section discusses the main challenges in realizing Raptor and explores its design and implementation. Deploying such a tool is non-trivial. First, there are multiple running tasks (e.g., network


*CoNEXT '21, December 7–10, 2021, Virtual Event, Germany*

ACM ISBN 978-1-4503-9098-9/21/12…$15.00
https://doi.org/10.1145/3485983.3493354

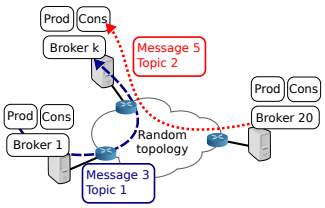


**Figure 2: Example scenario deployed using Raptor. Brokers k and 20 are the leaders for topics 1 and 2, respectively.**

switches, stream processors, data loggers) that must be accommodated on the same host. Second, some of these tasks must meet stringent performance requirements to work properly. For example, a broker replica must reply to periodic messages on time to not be considered outdated. Lastly, load can be significantly unbalanced, meaning a few tasks (e.g., the network control plane) may concentrate heavy load chunks.

Figure 1 shows Raptor's workflow. The tool takes as input: i) a set of actors (i.e., data producers and stream processors/consumers) specified by the application developer; ii) necessary configuration parameters (e.g., number of brokers, event topics and replicas) for setting up the underlying event streaming platform; and iii) a desired network topology to host the whole stream processing system. Raptor instantiates the specified topology using a network emulator. As part of this process, our tool provides several parameters that can be systematically tuned to model various operational conditions, including different routing and broker placement algorithms as well as failure profiles. Once the network is set, Raptor starts its event streaming platform and subsequently the user-specified producers/stream processors. Note that each of these actors run as an independent process which enables them to be balanced and prioritized among multiple cores. In addition, Raptor also triggers a series of monitoring tasks that are responsible for logging relevant information from both the network and the application perspective (e.g., bandwidth measurements, timestamped events). Finally, a visualization module presents to the user a rich set of statistics, such as per-port throughput and data synchronization latency (i.e., the time it takes for all subscribers to receive a message once it is published).

**Implementation.** We implemented Raptor on top of Mininet 2.3.0 and Kafka 2.8.0. Our system comprises approximately 1K lines of Shell and Python code. We use Networkx 2.5.1 and Matplotlib 3.3.4 to parse topology specifications (expressed as GraphML files) and present data visualizations to the user, respectively. The emulated network is proactively configured using a lightweight switch control daemon (based on ovs-ofctl) to bound the control plane overhead.

## 3 USE CASE

To demonstrate Raptor's benefits, we describe how it can be used to easily prototype a reasonably large stream processing system and test the latter's behavior under a variety of operational conditions. For that, we consider a geo-distributed scenario where multiple brokers (20 in our case) are placed on different locations. Each location also hosts a producer and a consumer that connect to their local broker for streaming data (see Figure 2). Every producer generates

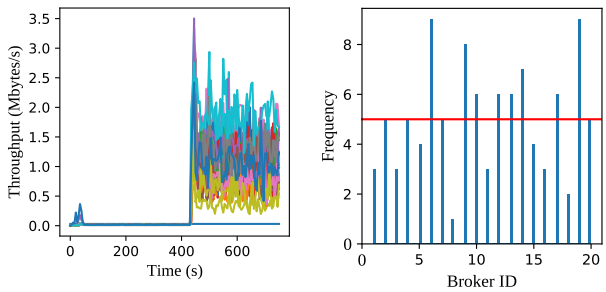


**Figure 3: (a) Receiving throughput at network access ports. First 400 seconds reflect the system warm up. (b) Distribution of topic leadership among brokers.**

data at a fixed rate of 30 KBytes/s and publishes it uniformly over 100 topics[1]. We assume a random topology where all links have the same capacity (1 Gbps) for simplicity.

Raptor parses this topology and automatically deploys the specified distributed stream processing application over the emulated network. Note that all network configuration details (e.g., addressing, routing policy) are transparent to the user, though can be manually set if needed. As part of Raptor's statistics, Figure 3a depicts the overall throughput of all network access ports (i.e., switch ports connected to end hosts) along time. We can see that even though traffic should be uniformly distributed among brokers as Kafka tries to balance topic leaderships, in practice bandwidth demands vary more than 4x among access links. That is mainly due to inefficiencies from the Kafka's load balancing mechanism.

To confirm the above hypothesis, we parsed Raptor's logs to identify the leading broker for each topic in our experiment. Figure 3b shows a histogram of the topic leadership distribution among brokers. The red bar illustrates the ideal scenario, where all brokers lead the same amount of topics, i.e., 5. As we can observe, the leadership frequency varies significantly among brokers, ranging from 1 to 9 and evidencing a strong unbalancing effect. These results show how Raptor can be used to improve the efficiency, e.g., by testing better load balancing mechanisms, of stream processing applications.

## REFERENCES

[1] Apache. 2021. Apache Kafka. https://kafka.apache.org/ Last accessed 30 September 2021.
[2] Yue Chen, Zhida Chen, Gao Cong, Ahmed R. Mahmood, and Walid G. Aref. 2020. SSTD: a distributed system on streaming spatio-textual data. *VLDB '20: Proceedings of the 2020 International Conference on Very Large Data Bases* (Aug. 2020), 2284–2296. https://doi.org/10.14778/3407790.3407825
[3] Sebastian Frischbier, Mario Paic, Alexander Echler, and Christian Roth. 2019. A Real-World Distributed Infrastructure for Processing Financial Data at Scale. *DEBS '19: Proceedings of the 13th ACM International Conference on Distributed and Event-based Systems* (June 2019), 254–255. https://doi.org/10.1145/3328905.3332513
[4] Ananad Jayarajan, Kimberly Hau, Andrew Goodwin, and Gennady Pekhimenko. 2021. LifeStream: a high-performance stream processing engine for periodic streams. *ASPLOS 2021: Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems* (April 2021), 107–122. https://doi.org/10.1145/3445814.3446725
[5] Mininet. 2021. http://mininet.org/. Accessed: 2021-10-06.

[1]Kafka always directs a producer/consumer to the topic leader while trying to balance leaderships among brokers.