Publish/Subscribe for Mobile Applications using Shared Dictionary Compression

Christoph Doblander, Kaiwen Zhang, Hans-Arno Jacobsen
Chair of Application and Middleware Systems
Technische Universität München
Email: {doblande, kzhang, jacobsen}@cs.tum.de

***Abstract*—Publish/Subscribe is known as a scalable and efficient data dissemination mechanism. In a mobile environment, there is an added challenge for the pub/sub system to economize mobile bandwidth, which is especially precious in areas not well covered by mobile providers. While well-known compression methods such as GZip or Deflate are generally useful in such situations, we propose using Shared Dictionary Compression (SDC) to achieve a greater level of bandwidth efficiency. SDC requires a dictionary, generated upfront, to be shared between two communicating peers before it can be used. We propose a design where brokers forming the pub/sub overlay can be in charge of generating and propagating the shared dictionary. Our solution employs an adaptive algorithm, executed at the brokers, which creates and maintains the dictionaries over time. With this approach, it is possible to reduce the required bandwidth by up to 88% including the introduced dictionary overhead. Our demo shows this approach applied to a smartphone application communicating with a publish/subscribe broker using the MQTT protocol.**

I. Introduction

In mobile applications, a pub/sub middleware [5] can be used for scalable and efficient dissemination of event notifications between the back-end infrastructure and the smartphone applications. As an example, the Facebook Messenger application uses MQTT [1], a pub/sub protocol, to communicate with the back-end servers. However, mobile phone data connections are often metered and the available bandwidth in rural areas tends to be lower than in city centers. Therefore, reducing the bandwidth consumption of pub/sub mobile applications translates into cost savings and allows their use even in areas where bandwidth is limited.

In this demo, we propose the use of Shared Dictionary Compression (SDC) in pub/sub in order to reduce the size of notifications to be sent to the mobile clients [4]. The approach extends the traditional pub/sub design by creating the role of *sampling broker*, which is responsible for sampling notifications, creating dictionaries, maintaining the dictionary over time and spreading the dictionary in the overlay network. Figure 1 shows an example of a topology which includes the sampling broker SB_1 which spreads a new dictionary through the overlay to all publishers P_{1-2} and subscribers S_{1-6}.

Our demo focuses on the implementation of a *sampling broker* by extending an existing pub/sub system using the MQTT [3] protocol. The sampling broker runs an adaptive algorithm which monitors the compression ratio and creates new dictionaries. Since disseminating a new dictionary introduces overhead, the algorithm decides to publish a dictionary only when it is beneficial. Periodic maintenance of the dictionary is needed since the content of the notifications changes over time. This can make a dictionary obsolete and reduce the bandwidth savings. In this case, the adaptive algorithm measures the impact on bandwidth reduction using a sample of the latest notifications. Then, a new dictionary is spread over the broker overlay if the current bandwidth consumption can be further reduced.

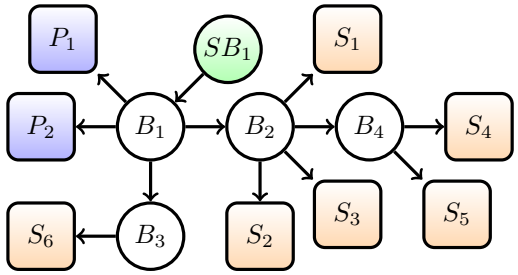

Fig. 1. Dictionary propagation from a sampling broker

II. Ride-sharing application

In the demonstration, we show this approach applied to a real world use case, which is derived from ride-sharing smartphone applications. Users are interested in getting real-time notifications of events around them shown on a map. To demonstrate this use case, we take the DEBS 2015 Grand Challenge dataset [6], a geospatial dataset of all NYC taxi trips for a year, and stream it to a smartphone.

III. Results of the use case

Listing 1. DEBS15-CSV

```
07290D3599E7A0D62097A346EFCC1FB5, E7750A37CAB07D0DFF0AF7E3573AC141
    ,2013 01 0100:00:00, 2013 01 01 00:02:00, 120, 0.44, 73.956528, 40.716976,
    73.962440, 40.715008, CSH, 3.50, 0.50, 0.50, 0.00, 0.00, 4.50
...
```

Listing 1 shows an example notification in CSV format. It includes the timestamp, geographic position, and other numerical attributes like fare, tips, etc. To demonstrate the effectiveness of the approach with different data formats, the demo application provides the ability to subscribe to same events in either XML, JSON, CSV, or Google Protobuf [2]. Table I shows the theoretical bandwidth reduction of our SDC approach compared to Deflate. The results also include the

Format	Deflate (%)	SDC (%)	SDC (avg bytes)
CSV (org)	34.5	62.0	72.75
XML	53.3	88.0	85.18
JSON	45.4	84.2	83.99
Protobuf	8.0	52.2	83.64

TABLE I
BANDWIDTH REDUCTION: SDC VS. DEFLATE

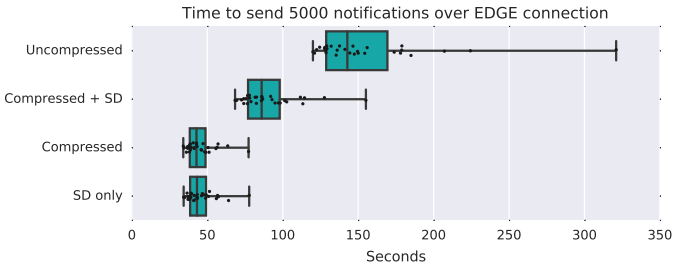

Fig. 2. Throughput: XML format

one time overhead of exchanging the dictionary. Surprisingly, even small binary notifications defined in Protobuf can be compressed by half using our approach. This is because SDC is able to leverage patterns found across Protobuf notifications to compress efficiently. Another interesting observation is that the final compressed size of notifications across formats is nearly the same, with at most 88% reduction for XML. Therefore, the developer can choose the notification format which is most practical or convenient without worrying about data size.

Figure 2 shows how long it takes to send a fixed volume of 5000 notifications when disseminated over a real world EDGE connection using our extended MQTT broker. The experiment is conducted by setting an Android phone to use an EDGE connection only while notifications are sent from a web server. Receiving the Shared Dictionary (SD) is a one-time cost upfront to use SDC. Once the SD is shared with the subscriber, the 5000 notifications can be sent $3x$ faster. The graph also shows that the cost of transmitting the dictionary is recouped after sending 5000 compressed notifications.

IV. IMPLEMENTATION AND ANDROID APPLICATION

We implemented the approach on top of a MQTT broker. The role of the sampling broker operates as an additional thread. A key-value store is employed to store active dictionaries. A newly joined publisher or subscriber can acquire the currently active dictionaries using an RPC mechanism. Figure 3 shows the components of the demo, which consists of several publishers emitting the ride-sharing events in different formats, the sampling broker, and the subscribers as smartphone applications. Screenshot 4 shows the Android application. The heat map displays ongoing events on the map. The progress bar shows how many notification per second are received. The buttons allow the client to subscribe to notifications in different data formats with or without compression in order to compare their relative performance.

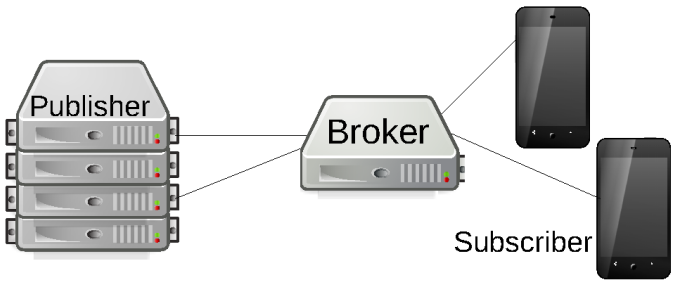

Fig. 3. Architecture of the demo

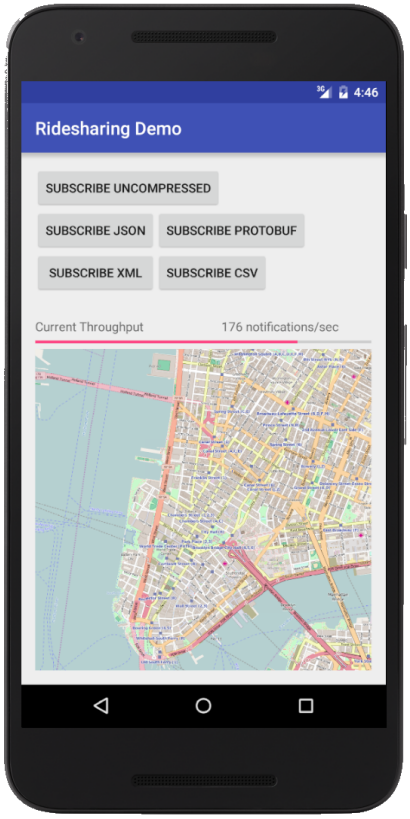

Fig. 4. Screenshot of the Android application

REFERENCES

[1] Facebook Messenger. https://www.facebook.com/notes/10150259350998920.

[2] Google Protocol Buffers. https://developers.google.com/protocol-buffers/.

[3] MQTT. http://mqtt.org/.

[4] C. Doblander, T. Ghinaiya, K. Zhang, and H.-A. Jacobsen. Shared Dictionary Compression in Publish/Subscribe. In *Proceedings of the 10th ACM International Conference on Distributed Event-Based Systems*, DEBS '16, 2016.

[5] H.-A. Jacobsen, A. Cheung, G. Li, B. Maniymaran, V. Muthusamy, and R. S. Kazemzadeh. The PADRES Publish/Subscribe System. In *Principles and Applications of Distributed Event-Based Systems*, pages 164–205. IGI Global, 2010.

[6] Z. Jerzak and H. Ziekow. The DEBS 2015 Grand Challenge. In *Proceedings of the 9th ACM International Conference on Distributed Event-Based Systems*, DEBS '15, pages 266–268, New York, NY, USA, 2015. ACM.