Keyword-Based Mobile Application Sharing

Ioannis Psaras*, Sergi Reñé*, Kostantinos V. Katsaros‡, Nikolaos Bezirgiannidis†,
Vasilis Sourlas*, Sotiris Diamantopoulos†, Ioannis Komnios†, Vassilis Tsaoussidis†, George Pavlou*

*Dept. of Electrical & Electronic Engineering, University College London, London, UK
Email: {i.psaras, s.rene, k.katsaros, v.sourlas, g.pavlou}@ucl.ac.uk
†Dept. of Electrical & Computer Engineering, Democritus University of Thrace, Xanthi, Greece
Email: {nbezirgi, sdiaman, ikomnios, vtsaousi}@ee.duth.gr
‡Intracom Telecom, Athens, Greece
Email: konkat@intracom-telecom.com

***Abstract*—The advent and wide adoption of smartphones in the second half of '00s has completely changed our everyday mobile computing experience. Tens of applications are being introduced every day in the application markets. Given the technology progress and the fact that mobile devices are becoming strong computing devices, mobile *applications* are expected to follow suit and become computation-heavy, bandwidth-hungry and latency-sensitive. In this paper, we introduce a new mobile computing paradigm to alleviate some of the network stress that mobile applications are already putting into the network, *e.g.*, in case of crowded areas and events, where the mobile network effectively collapses. According to this paradigm, *users can share the applications that they have on their mobile devices with nearby users* that want access to *processed* information, which their own applications cannot provide. In a sense, then, the client application instance is also acting as a server instance in order to serve requests from nearby users. A representative example is a route-finder application in a busy station, airport, stadium or festival, or a gaming application onboard a flight. Our paradigm builds on Information-Centric Networking (ICN) and uses *keyword-based requests* to discover shared applications in the vicinity.**

I. Introduction

Mobile computing is currently led by smartphones and is largely application-centric. Users increasingly rely on applications to gain access to information, *e.g.*, the top-100 applications in most popular application markets are responsible for almost 90% of the access time and 80% of the traffic volume [1]. Through applications, users normally gain access to *processed information*, *e.g.*, finding a route, searching for restaurants, getting personalised social networking or news feeds, *etc.*, instead of only asking for static content. Although modern mobile devices possess remarkable computing capabilities, the required resources for application processing are primarily provided by the cloud. Subsequently, using smart applications and the corresponding cloud-based service components, *e.g.*, Facebook, depends on the availability of Internet access, increasingly stressing the network infrastructure.

However, access to the cloud is in some cases not necessarily the best option, or not always possible. It is not uncommon the case where connectivity and access to the Internet is challenged in overcrowded areas (*e.g.*, airport lounges, festivals, stadiums, big conference-like events), due to equipment failure (*e.g.*, in case of natural disasters), or complete absence of available connectivity (*e.g.*, onboard a flight, or in a train while in tunnel), or even due to high roaming costs. It is important to understand, however, that in those cases, a very big proportion of the services that users are interested in do not actually require access to the global network, but are rather targeting non-personalised services or processed information related to the local event itself. For instance, while in a festival, users are more likely to be interested in finding information on local restaurants, train times or local maps, rather than requiring VPN connection to their work email. Although the situation can be quite different in a conference-style, business-oriented event, we argue that the demand for local services is still far from negligible [2]. In all cases, from a resource management point of view, dealing with demand for local services locally, increases the availability of Internet resources to those who do require access to remote services. In [3] the authors claim that enabling content-sharing between devices in sport events decreases bandwidth consumption by $\sim$50%.

To this end, several solutions have been proposed to share information between mobile devices bypassing the Internet infrastructure, *e.g.*, FireChat[1]. A significant body of work has focused on mobile ad-hoc networks, however inheriting the drawbacks of the underlying host-centric IP paradigm *i.e.*, location-identity coupling. Taking a data-centric approach, Haggle [4] first proposed a data-centric network oriented to sharing content in local mobile networks that enabled seamless network connectivity and application functionality in dynamic mobile environments, separating application logic from transport bindings so that applications can be communication-agnostic. Trying to overcome IP limitations, other proposals focused on the Information-Centric Networking (ICN) paradigm, *e.g.*, [5]. In [3], the authors propose Krowd to enable content sharing between users in crowded live events by realising a key-value store abstraction for applications, providing single-hop content discovery and sharing with the participation of local access points.

Other approaches have been based on Delay-Tolerant Networks (DTN), exploiting both its inherent capability to exchange data in opportunistic environments, and its in-network storage functionality. For instance, a DTN-based content storage and retrieval platform is proposed in [6], enabling applications to make caching and forwarding decisions. In [7] maps of disaster areas are generated and shared over a distributed DTN-based computing system. Similarly, the Floating Content [8]

[1] http://opengarden.com/firechat

concept leverages ad-hoc communications among mobile users to share local information. According to [8], message and information replication is limited in time and space.

The proposed solutions so far aim at either enabling IP-based connectivity in mobile environments, or supporting the generic, application-agnostic exchange of content and computations often employing ICN primitives, *e.g.*, name-based routing and forwarding [9]. Named Function Networking (NFN) [10] extends the resolution-by-name ICN primitives providing in-network data computations, but without enabling application sharing in mobile environments. Last, but not least, the recent trend towards *"distributed edge-mobile or fog computing"* is pushing application logic closer to the end user [11]. Although still in its early days, the concept of fog computing attempts to bring computation and processing of information (*i.e.*, the cloud) closer to the end-user. The main benefit of this paradigm is more efficient use of resources and reduced response latency.

In this article, we take a step further from content sharing, host-centric communications and fog computing and focus on the prevailing application-centric computation and communication model. The proposed framework explicitly enables access to the desired *processed and non-personalised* information through the concept of *application sharing*, effectively leveraging on a *pool of application resources*. Namely, we leverage application-centrism to facilitate information discovery through application-driven and application-defined, hierarchical namespaces. Given the ad-hoc nature of the proposed computation framework, our approach further extends these namespaces by introducing the concept of *keywords*. This enables the description, discovery and retrieval of *processed* information, further supporting variable accuracy results, instead of only exact matches, *e.g.*, a search result that does not contain all search terms. Note that the invocation of remote processing (in co-located smartphone or WiFi AP devices) is central to our framework, as opposed to previous work on retrieving static content from nearby devices. Our *keyword-based mobile application sharing framework (KEBAPP)*, manages connectivity in an application-centric way, *i.e.*, coupling connectivity options and opportunities to applications and their namespaces. KEBAPP extends existing ICN primitives, namely CCN/NDN [12], thus resulting in a generic solution across different applications and overcoming the pitfalls of IP.

II. The KEBAPP Framework

We present KEBAPP, a new application-centric information sharing framework oriented to support opportunistic computing between mobile devices. Our approach targets scenarios where large numbers of mobile devices are co-located, presenting the opportunity for localised, collective computing with a special focus on *application sharing and information processing*. In this context, KEBAPP employs application-centrism to facilitate and enable (i) *the exchange of processed information, in contrast to merely static content*, and (ii) *the discovery and delivery of information to satisfy user interests*.

Figure 1, presents the structure of a KEBAPP-enabled host. KEBAPP provides a new layer between the application and the link layers exhibiting three major design features: (i) *application-centric naming, where applications share common name-spaces and further support the use of keywords* (Section II-A), (ii) *application-centric connectivity management (KEBAPP WiFi Manager), where applications manage connectivity by defining and/or joining WiFi broadcast domains* (Section II-B), and (iii) *information-centric forwarding, extending CCN/NDN primitives* (Section II-C).

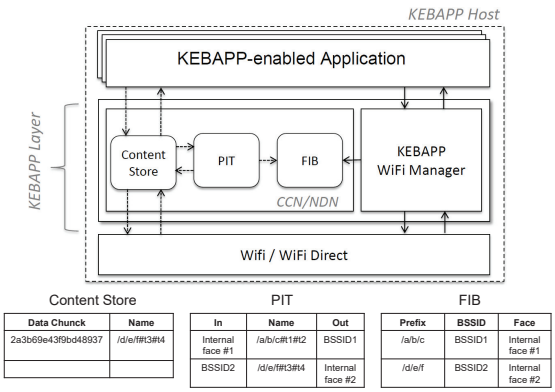


Fig. 1. KEBAPP-enabled host

A. Naming

The discovery and invocation of services/applications in the networking vicinity of a user builds on a naming scheme that enables the fine-grained description of the desired processed information. To this end, KEBAPP builds on the observation that mobile computing is largely application-centric, *i.e.*, users tend to access information using purpose-built applications, rather than web-browsers. Applications present the important characteristics of inherently: (i) supporting the structuring of the namespace within their semantic context and (ii) being used for computation, enabling the (lightweight) processing of information, *e.g.*, searching, sorting data or computing a route.

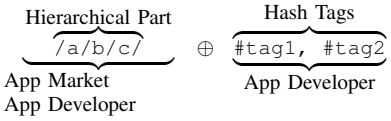
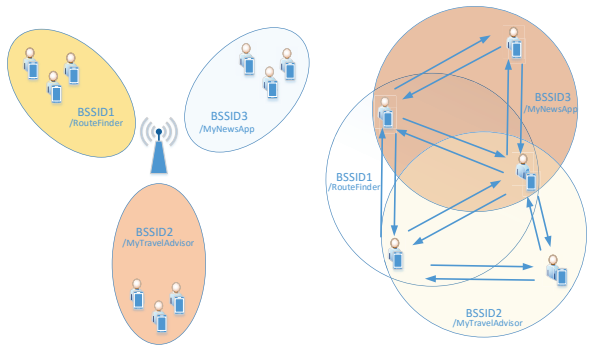


Fig. 2. Keyword-based Names

Taking these features into account, KEBAPP names are composed of two main parts (see Figure 2):

Fixed Hierarchical Part. It follows the hierarchical naming scheme of CCN/NDN and its purpose is to guarantee compatibility between instances of the same or different services/applications. Application developers can define their own hierarchical namespaces, enabling communication between different instances of the same (or similar) application, such as categories in a news application., *e.g.*, `/NewsApp/politics/international`.

Moreover, application developers can also define suffixes corresponding to specific functionalities within their applications (in addition to static content), enabling this way the sharing of computation, *e.g.*, the name `/MyTravelAdvisor/Top10Restaurants` is used to identify the list of the top-10 restaurants in a certain area.



(a) AP-assisted scenario (b) WiFi Direct scenario

Fig. 3. Connectivity Options

According to our initial design the hierarchical part of the name will have to be an exact (longest prefix) match in order for a request to be served. It is noted though, that this matching is performed by the KEBAPP layer, with the user simply interacting with the application GUI, *i.e.*, users need not be aware of the exact naming conventions.

Hashtags. The second part of the name comprises of hashtag-like free keywords, which the application developer can add to the application. The exact semantics of the hashtags depend on whether the fixed hierarchical part of the name corresponds to static content or an application function(ality). In the former case, these keywords are used to semantically annotate the static content. This feature enables the partial matching of requests, in contrast with the longest prefix match used in NDN, with the available cache or routing/forwarding entries, *i.e.*, given an exact match in the fixed hierarchical part of the name, hashtags can be used to support approximate matching, in turn enabling the search of information in nearby devices.

When the fixed part of the name identifies a certain application function(ality), the hashtag part of the name enables the passing of adequate parameters. In the aforementioned example of the *MyTravelAdvisor* application, the complete name included in a user request can have the fixed hierarchical part `/MyTravelAdvisor/Top10Restaurants` and the hashtags `#userrating, #London, #indian` indicating that the user is interested in the top-10 of the indian restaurants in London, according to users ratings. The submission of hashtag values is guided by the application GUI and can include both predefined value ranges, *e.g.*, the sorting criteria for the top-10 restaurants, and free text fields, *e.g.*, a user requests `/MyNewsApp/politics/search #Syria #negotiations` to use the search function of *MyNewsApp* and find anything related to negotiations for Syria.

B. Connectivity Management

Connectivity management plays a vital role in KEBAPP. In this work we focus on WiFi-enabled (IEEE 802.11) connectivity. This also includes WiFi Direct, which enables mobile devices to act as access points (APs) by forming communication groups. In KEBAPP, we propose the creation and use of 802.11 broadcast domains for the support of particular applications, *i.e.*, KEBAPP-enabled hosts or APs advertise one or more Basic Service Set(s) (BSSs) for the support of one or more application(s). The creation of application-specific BSSs aims at enabling mobile devices to connect only when their counterparts support the same application and/or namespace. Within a BSS, hosts communicate employing CCN/NDN primitives, as described in Section II-C.

The advertising AP or host, through a WiFi Direct Group, acts as a mediator to connect different users willing to share the same application in a single broadcast domain. In the case of APs, functionalities such as access control, association, encryption, *etc.*, can be supported without imposing computation and/or battery overheads to mobile devices. Note however, that APs in this case need not provide access to the Internet. Figure 3a represents an AP-assisted scenario where different users share different applications.

The creation of an application-specific BSS requires the ability of mobile devices to identify the mapping between the BSS and the corresponding application. The recently announced WiFi Neighbour Awareness Networking (NAN) protocol [13] can support this requirement. Namely, WiFi NAN supports a low energy consumption device discovery mechanism enhanced with publish/subscribe primitives that can serve to retrieve what application is available in a certain BSS. Other technical approaches are also possible, *e.g.*, employing the Access Network Query Protocol (ANQP) of IEEE 802.11u or using pre-defined SSIDs. It is noted that a device can be connected to more than one BSSs at the same time (*e.g.*, [14]), thus acquiring or providing information across several applications (Figure 3b).

C. Forwarding Operation

The basic forwarding operation of a KEBAPP node is a modified version of Named Data Networking (NDN) architecture [12]. The KEBAPP modifications aim at reflecting the forwarding of messages within the various BSSs a node may participate. As explained in the following, since we consider single-hop broadcasting domains, forwarding decisions lead to either the broadcasting of a message in the BSS or its delivery to a local application instance. As such, broadcast domains are considered as (inter)faces of a KEBAPP node.[2]

The KEBAPP forwarding scheme, similarly to NDN, has three main data structures: FIB (Forwarding Information Base), CS (Content Store) and PIT (Pending Interest Table). The FIB is used to forward Interest packets toward potential sources of matching data. The WiFi manager (see Figure 1) populates the FIB table with the name prefixes (hierarchical part of the name scheme detailed in Section II-A) advertised by the wireless networks in the vicinity, *e.g.*, through WiFi NAN. As in NDN, a FIB entry comprises the name prefix and a list of output face(s). In KEBAPP, the latter list (of output faces) includes the Basic Service Set IDentifiers (BSSIDs) advertised by other nodes. Moreover, when a KEBAPP node acts as information producer, providing information to other nodes, the output face list is further augmented with the `Internal_Face`, which enables a node to forward an Interest message to the local instance of the application (*i.e.*, the device that acts as a server).

The second main data structure, the CS, is responsible of caching the information requested by the users, providing fast

[2]We focus on the KEBAPP functionality; details relating to the coexistence of KEBAPP with original NDN are out of the scope of this paper.

fetching for popular information and avoiding to recompute information already requested by other users. Replacement policies of the CS is out of the scope of this paper. We note, however, that in the context of KEBAPP, caching happens in terms of *processed* information, as opposed to static content and therefore, cache hits happen only in case of requests with similar processing requirements (*e.g.*, restaurants within the same geographical boundaries).

Finally, the third data structure of a KEBAPP-enabled node, the PIT, keeps track of Interests, containing the full name, forwarded to any BSS. KEBAPP keeps a PIT entry for every application request. Depending on whether an Interest message comes from the local application instance or another node in the corresponding BSS, the Requesting Faces list contains a handle to the local application instance (*i.e.*, the `Internal_Face`) or the BSSID of the corresponding BSS. In order to support delay tolerant communications, KEBAPP extends NDN's functionality, by allowing the creation of PIT entries even when no suitable destination, *i.e.*, forwarding entry, has been found for the Interest message. At the same time, a PIT entry is extended to further indicate the Destination Face, *i.e.*, the BSSID, it has been broadcast to or the corresponding `Internal_Face`. This serves the purpose of (re-)issuing Interest messages upon the discovery of a BSS that is associated with a matching name (in the FIB).

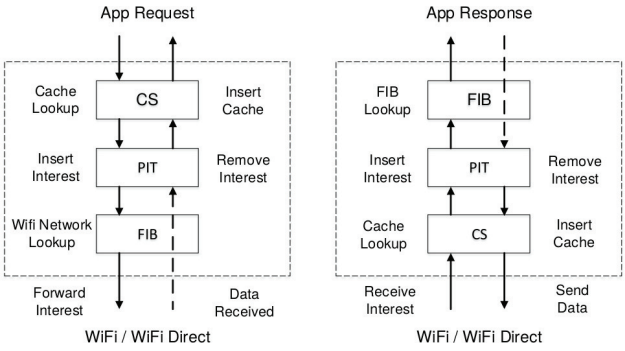


Fig. 4. Forwarding operations

Figure 4 provides a representation of the KEBAPP packet forwarding engine. In the following, we detail the operation of KEBAPP framework for an *information requester*:

1) The application requesting for information creates a new Interest.
2) The application looks for the information in the local CS. If the information exists locally, the data is sent to the application.
3) If the information is not found in the CS, the KEBAPP layer inserts an entry in the PIT (<`Internal_Face`, `name_prefix + keyword_list`, *`null`*>). As in NDN, we use the term "Internal Face" to point to the local application involved in the transaction (either as requester or as provider).
4) The KEBAPP (network) layer checks if there is a BSSID entry in the FIB matching the `name_prefix` of the PIT.[3] If an entry for the requested name prefix exists, the WiFi manager connects the WiFi interface to the BSSID in the FIB and broadcasts the Interest message with a corresponding time-out value.
5) Each time a new FIB entry is added because a new prefix name is discovered on a new BSS (*e.g.*, through WiFi NAN), the KEBAPP layer checks if a *pending* PIT entry for this prefix exists. As mentioned above, this corresponds to PIT entries created for Interest messages that could not be forwarded. In case an entry exists, the Interest is sent through the recently added BSSID, and the entry is updated with the BSSID value.
6) When a response is received with the information requested, the KEBAPP layer looks for the internal face that points to the application in the corresponding PIT entry and fordwards the response to it. The PIT entry is removed and the information requested is cached.

[3]Note that a local "Internal_Face" will never be used since this is a local request.

Next, we describe the operation of the KEBAPP framework for an *information provider*:

1) The user receives an Interest through the interface connected to a certain BSS related to an application.
2) The KEBAPP layer checks the CS for matching entries.
3) In case there is no entry in the CS matching the Interest, a PIT entry is first created. This entry allows the provider device to serve multiple, concurrently arriving, identical requests with a single message, *i.e.*, applying multicast, as in original CCN/NDN. In this case, the Requesting Face list of the entry includes the BSSID of the current BSS. Subsequently, the FIB table is looked up and the `Internal_Face` is used to forward the Interest message to the corresponding application. For completeness, the `Internal_Face` is also added to the PIT entry as an output face.
4) The response from the application is cached in the CS and sent back to the broadcast domain indicated by the BSSID value of the local PIT entry, which is subsequently removed.

III. Preliminary Results

A. Use Case: RouteFinder App

For the evaluation of the proposed framework, we consider a *RouteFinder* application that provides information on train lines and their respective schedule, as well as real-time information regarding delays, closed stations etc. When realising a *RouteFinder* application, the KEBAPP framework will have to deal with one of the two following cases. The first case is when some other device (either a client device or an AP) has previously setup a BSS, using WiFi in the case of an AP, or a WiFi Direct group in the case of a client device, advertising the corresponding application. The second case, is when no other BSS advertising the service required can be detected in the vicinity. In this second case, the user sets up a new BSS with the service *required* and waits for other users willing to connect to the BSS to share the requested application/service (see Sec. II-C - pending PIT entries).

B. Evaluation framework

For the evaluation of the viability and the benefits of the proposed framework, we study a proof-of-concept performance of the KEBAPP *RouteFinder* app using a custom-built Java simulator. To realistically simulate user mobility, we employ a relevant mobility trace, which corresponds to 3300 users at a subway station in downtown Stockholm[4].

In particular, we assume that only a subset of the 3300 users that enter the station within one hour are KEBAPP users (which reflects the KEBAPP penetration rate) and we examine to what extent route calculation requests are successfully responded by apps in other user devices. We randomly select a subset of the trace nodes as the KEBAPP users, and randomly generate one route calculation request per user during their short stay in the station. In Fig. 5, we assume that 10 percent of the overall user population are KEBAPP users and show that, on average, only 18 KEBAPP users co-exist in the subway station at any given moment. We assume that not all KEBAPP users can perform the required computations (*i.e.*, possess the *RouteFinder* app), so a request can be successfully responded by only a small percentage (1%, 5%, or 10%) of the other KEBAPP users, which are also selected at random. We also assume that if two users are in the station within the same slot, they are also within range of each other. We elaborate and relax this assumption later on.

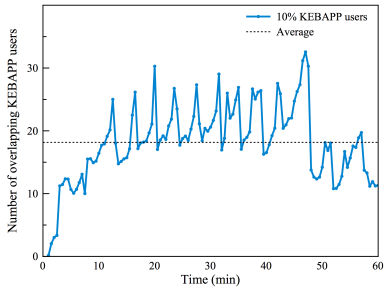


Fig. 5. Number of co-located KEBAPP users

All sets of simulations are repeated 1000 times. We measure the successful responses of route calculation requests in terms of response probability and response time. In particular, we measure: *i)* the *average Response Ratio (RR)*, *i.e.*, the fraction of the total generated requests that receive a successful reply, *ii)* the *average First Response Time (FRT)*, which is the average time between issuing a request and receiving the first reply (for those requests that are successfully responded), and *iii)* the *average Response Time (RT)*, which is the average response time considering all successful responses. In order to incorporate application compute time and protocol-specific delays in our simulation, in the last set of experiments we induce a transmission and computation delay to each response and study the corresponding *Failure Rate (FR)*. The Failure Rate is the fraction of the total requests that cannot be successfully responded due to the movement of nodes (*i.e.*, the responding node received the request but moved out of range of the requesting node immediately after).

[4]Trace was obtained from http://crawdad.org/kth/walkers/20140505/

C. Evaluation results

Our evaluation results show that, as expected, the response ratio increases with the KEBAPP-enabled application penetration rate, as well as with the percentage of sharing users. Overall, we see that even with a 5% penetration rate, (*i.e.*, when only 165 out of the 3300 commuters are KEBAPP users), the KEBAPP users have a chance to get a successful response. As shown in Fig. 6, the lowest response ratio of 16.8% is achieved when only 1% of the 165 KEBAPP users are simultaneously present in the station and can successfully respond to the users' requests. The probability of a successful response gets significantly higher when more users are willing to share their resources; this probability is increased from 16.8% with only one user sharing resources to 76.7% with 10% of users sharing resources out of 165 KEBAPP users - see first set of bars in Fig. 6. As the number of KEBAPP-users increases, the response ratio approaches 100%.

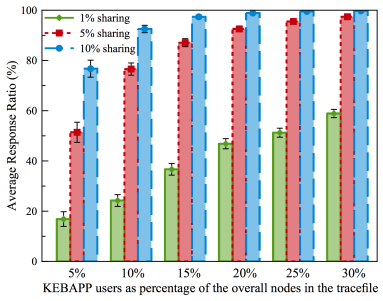


Fig. 6. Average Response Ratio for increasing number of KEBAPP users

As far as the average response time is concerned we notice, in Fig. 7, that first response time decreases as the number of users that are sharing resources increases. In particular, the maximum average delay is observed for the lowest sharing ratio of 1%, and it spans from 17-29 seconds. For more realistic sharing ratios of around 10%, we observe a user request can almost immediately find a match (*i.e.*, within a couple of seconds). Note that in a future scenario where mobile application sharing is widespread, the response rates and times from other devices could be comparable to nowadays cloud-based services. This is a very encouraging result, which naturally leads us to investigate the failure rates of connections and the failure to complete requests for processed information.

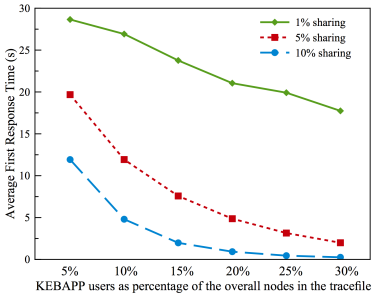


Fig. 7. First Response Time for increasing number of KEBAPP users

In order to evaluate the proposed framework in a more realistic setup, we add one more feature to our simulation

setup. In particular, we add extra processing delay between issuing a request and receiving a response. This delay accounts for signalling, computation and transmission between the two communicating nodes. Recall our assumption that if two nodes are inside the station during the same time period, then they are also within range of each other. This might not always be realistic, as stations are normally spread across many floors, covering areas in the order of a few hundred square meters. That said, the co-ordination between requesting and responding devices might need to be handled by some central controller or WiFi access point. In turn, such co-ordination would inevitably add extra signalling overhead which we try to incorporate in our simulations in this last scenario.

In order to realise the above concerns, we induce 1-5 secs of extra delay to each response and study the corresponding *Failure Rate (FR)* when 5% of KEBAPP users possess the application in question and can therefore reply to incoming requests. Given the high mobility of users in such an environment, the failure rate denotes the percentage of users that have moved out of the station by the time a response is back. As shown in Fig. 8, failure rates range between 0.5% to 3.5% of all requests. As expected, failure rates increase as the extra delay to respond to incoming requests increases, *i.e.*, failure rate is less than 1% when the extra processing delay is 1s and increases to approximately 3.5% when the extra delay is 5s. Although one would expect the failure rate to decrease as the percentage of KEBAPP users increases (on the x-axis of Fig. 8), we note that as the KEBAPP users increase, so does the incoming requests. Therefore, the stability of the result in Fig. 8 proves the stability of the system over time.

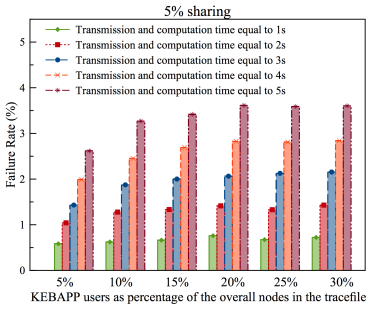


Fig. 8. Failure Rate for increasing number of KEBAPP users

IV. Conclusions

We have introduced the concept of *Keyword-Based Mobile Application Sharing* (KEBAPP), according to which applications installed in users' devices act both as clients (requesting for some service) and as servers (responding to incoming requests). In that sense, smartphone apps of collocated devices act as a pool of applications available to all users in the area. In turn, clients can make use of applications in nearby devices without the need to necessarily reach out to the cloud to retrieve processed information. KEBAPP builds on ICN principles and forms requests based on keywords and hashtags in order to invoke computation in nearby devices. As a last step, *processed information* is returned to the requesting client.

In realising such a framework, a number of issues still remain open. Such issues include backward compatibility studies (*i.e.*, provision for as little disruption to the current application market as possible), authentication (*i.e.*, who authenticates and how that the responding application is the application that it claims it is) and security against attacks (*i.e.*, how to avoid users flooding other devices with bogus requests). Despite the long list of issues, we believe that KEBAPP is a promising direction towards edge-computing resource pooling.

Acknowledgments

This work has been supported by the EC H2020 UMOBILE project (GA no. 645124) and the EPSRC INSP Early Career Fellowship (no. EP/M003787/1). V. Sourlas is supported by the European Commission through the FP7-PEOPLE-IEF INTENT project, (GA no. 628360).

References

[1] Q. Xu, J. Erman, A. Gerber, Z. Mao, J. Pang, and S. Venkataraman, "Identifying diverse usage behaviors of smartphone apps," in *ACM ICN'11*, pp. 329–344.

[2] I. Wakeman, S. Naicken, J. Rimmer, D. Chalmers, and C. Fisher, "The fans united will always be connected: building a practical dtn in a football stadium," in *ADHOCNETS 2013*.

[3] U. Drolia, N. Mickulicz, R. Gandhi, and P. Narasimhan, "Krowd: A key-value store for crowded venues," in *MobiArch 2015*, pp. 20–25.

[4] J. Scott, P. Hui, J. Crowcroft, and C. Diot, "Haggle: A networking architecture designed around mobile users," in *WONS 2006)*.

[5] C. Anastasiades, T. Braun, and V. Siris, "Information-centric networking in mobile and opportunistic networks," in *Wireless Networking for Moving Objects*, pp. 14–30, 2014.

[6] J. Ott and M. J. Pitkanen, "Dtn-based content storage and retrieval," in *IEEE WoWMoM 2007*.

[7] E. Trono, Y. Arakawa, M. Tamai, and K. Yasumoto, "Dtn mapex: Disaster area mapping through distributed computing over a delay tolerant network," in *ICMU 2015*.

[8] J. Ott, E. Hyytia, P. Lassila, T. Vaegs, and J. Kangasharju, "Floating content: Information sharing in urban areas," in *IEEE PerCom 2011*.

[9] G. Xylomenos and et al., "A survey of information-centric networking research," *Communications Surveys Tutorials, IEEE*, vol. 16, no. 2, 2014.

[10] M. Sifalakis, B. Kohler, C. Scherb, and C. Tschudin, "An information centric network for computing the distribution of computations," in *ACM ICN 2014*.

[11] Y. Coady, O. Hohlfeld, J. Kempf, R. McGeer, and S. Schmid, "Distributed cloud computing: Applications, status quo, and challenges," *SIGCOMM CCR*, vol. 45, Apr. 2015.

[12] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard, "Networking named content," in *ACM CoNEXT'09*.

[13] D. Camps-Mur and et al., "Enabling always on service discovery: Wifi neighbor awareness networking," *Wireless Communications, IEEE*, vol. 22, pp. 118–125, April 2015.

[14] A. J. Nicholson, S. Wolchok, and B. D. Noble, "Juggler: Virtual Networks for Fun and Profit," *IEEE Transactions on Mobile Computing*, vol. 9, pp. 31–43, Jan. 2010.