Surpassing Flow Fairness in a Mesh Network: How to Ensure Equity among End Users?

Sandip Chakraborty, Sukumar Nandi, Subhrendu Chattopadhyay
Department of Computer Science and Engineering,
Indian Institute of Technology Guwahati, Assam, India 781039
Email: {c.sandip, sukumar, subhrendu}@iitg.ernet.in

***Abstract*—Assuring user fairness in IEEE 802.11 wireless mesh network is challenging mainly for two reasons - first, the in-built unfair behavior of IEEE 802.11 contention based channel access in multi-hop forwarding, and second, the capacity degradation of mesh routers near a gateway due to the heavy traffic load. This paper studies the fundamental reason of fairness degradation in a mesh network, using a practical mesh testbed. The experimental results reveal that the flow fairness alone cannot solve this issue, because the near-gateway routers with high traffic load require more bandwidth for traffic forwarding. An improved channel access differentiation technique based on a load estimation strategy is proposed in this paper to reduce the capacity degradation of mesh routers, and to improve the fairness among end users. The proposed protocol is evaluated using results obtained from two different testbed setups. These results show that the proposed protocol improves the end-user fairness, as well as the average throughput in the network.**

***Keywords*-mesh; IEEE 802.11; fairness; capacity; flow control**

I. Introduction

IEEE 802.11 Wireless mesh network [1] has generated interests among researchers because of its capability of replacing the wired infrastructure by wireless backbone. Mesh backbone network is mainly comprised of wireless routers which have two functionalities - to act as the access point (AP) to client nodes, and to relay the traffic from other routers towards or from the gateway. One or more of the mesh routers also act as gateways to connect outside network.

A mesh network over IEEE 802.11 technology experiences severe problem of unfairness among end-users. IEEE 802.11 Enhanced Distributed Channel Access (EDCA) provides equal time share to every contending station when every station has sufficient data to transmit. Therefore, mesh routers with high traffic load (mainly routers near a gateway) suffer from the unavailability of sufficient bandwidth, and the performance degrades drastically for the end users who are multiple hops away. Though several proposals exist in literature to study fairness issues in multi-hop and mesh networks, such as [2]–[4] and their references, they primarily concentrate on end-to-end flow fairness. A second class of works studies fairness issues in channel access protocols, such as [5]–[7] and their references, that mainly focus on prioritizing the channel access of individual users, according to their traffic demand and the quality of service criteria. However, these works do not reveal the fundamental reasons behind the fairness degradation in a mesh network. Assuring end-to-end fairness cannot solve this issue alone, as the capacity of mesh routers near a gateway degrade severely at high traffic load. Therefore traffic control at intermediate routers is necessary to ensure equal transmission opportunity to every end-user.

In our previous paper [8], an improved fairness provisioning protocol has been proposed that estimates traffic load at every intermediate router. The channel access mechanism of every router is either prioritized or deprived based on the traffic load and the contention information in the neighborhood. Therefore, every router gets channel access proportional to its traffic load (called the *proportional fairness* criteria) that ensures traffic equality among all flows. The contention window (CW) of IEEE 802.11 EDCA has been tuned to prioritize or to deprive the channel access of individual routers. However, the load estimation strategy proposed earlier is based on the traffic demand of individual clients that varies with time according to application usage, and thus difficult to compute, account and manage at router level. Further the CW tuning mechanism considers only short-term effect that may result in sudden performance degradation when new flows join in the network, or existing flows terminate.

The objective of this paper is two-fold. The first objective is to study the unfair behavior of end-users in a mesh network even with the support of an end-to-end flow-fairness control protocol, such as fair queuing, and second the second objective is to provide an optimized implementation of the previously proposed protocol [8] with an improved efficiency. A practical testbed of mesh network has been used for this purpose. The major contributions of this paper are summarized as follows.

- The results obtained from a practical testbed are used to study the problem of fairness degradation in a mesh network even with the support of end-to-end flow fairness at transport layer. The study reveals that clients associated with the routers near the gateway reserve the full capacity of the network, as a consequence of which, the flows with large number of hops get starved.
- An improved protocol is proposed over [8] to support channel access based on the traffic load at individual routers. The load estimation strategy only relies on the router level information, such as the size of the interface queue, and therefore free from the time varying information, that may introduce stale results. Further, the CW tuning mechanism is optimized by considering long-term

effects.

- The earlier protocol proposed in [8] was evaluated using simulation results only. This paper extensively studies the performance of the proposed protocol using two different testbed setups. The performance of the proposed protocol is compared to the performance of EDCA MAC along with the transport layer fair queuing support.

II. Analysis of Unfairness in Mesh Network

This section analyzes the fundamental reasons of unfairness among end-users in a wireless mesh network, through the results obtained from experiments conducted using an 802.11b testbed in a multi-storied building.

A. Testbed Setup

An 8 node testbed, as shown in Fig. 1, is used where one node is selected for the gateway, and rest other nodes act as mesh routers. It can be noted that the terms AP and routers are used interchangeably, as mesh routers also act as an AP for clients. Each node is a Skiva Easyconnect RT001 N300 WiFi router with RaLink RT-3352 chipset [9]. The Ralink RT-3352 router on chip combines 802.11n draft compliant 2T2R MAC/BBP/PA/RF, a high performance 400MHz MIPS24KEc CPU core, a Gigabit Ethernet MAC, 5-ports integrated 10/100 Ethernet Swtich/PHY, 64MB of SDRAM and 32MB of Flash. The nodes of the testbed are distributed over the corridor of the hostel Brahmaputra at Indian Institute of Technology Guwahati campus. The nodes are located roughly 15-20 meters apart from each other. The dotted lines in Fig. 1 show the approximate connectivity among these mesh routers. The forwarding path is determined based on AODV routing protocol with *expected transmission count* (ETX) as the routing metric [10], so that routing paths are evenly distributed, and do to effect the MAC layer access protocol.

The MAC layer uses IEEE 802.11 EDCA with the support of the transport layer end-to-end flow fairness control protocol as proposed in [11]. The protocol proposed in [11] uses per-flow queuing to maintain and end-to-end bandwidth requirement, and controls the transport layer data rate so that fairness is assured. Though per-flow queuing is not scalable, as well as requires large amount of memory to implement, and several other protocols have been proposed in literature to overcome the shortcoming of per-flow queuing, it is known to perform better than other protocols if queue maintenance is possible [12]. Therefore, it is used for comparison purpose.

During the experiments, on an average 5 clients are associated with every router. The data communications are between clients and the outside network through the mesh gateway. Three batches of experiments are conducted. First, The clients have uploaded a large file of size 1 GB to a server which is connected to the outside network through the mesh gateway. To upload the file, Trivial File Transfer Protocol (TFTP) is used that uses UDP. Next, clients have uploaded the same file using File Transfer Protocol (FTP) that uses TCP. Finally, three different applications are randomly distributed among clients - website browsing including media streaming, TFTP and FTP. For all the experiments, the physical data rate of mesh routers is kept fixed at $6Mbps$. The MAC implementation is capable of sending and receiving raw 802.11 frames using the wireless network interface in monitor mode. The Ralink AP SDK 3.3.0.0 [9] has been used for the development purpose.

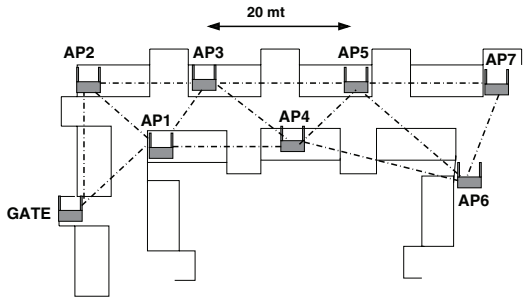


Fig. 1. Location of nodes in the 8-node testbed

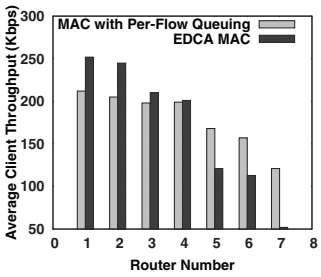


Fig. 2. Clients' avg UDP throughput per router

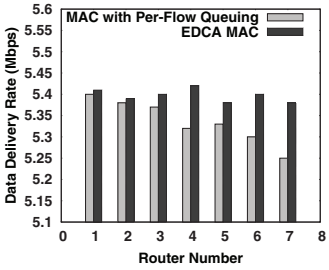


Fig. 3. Per router packet delivery rate

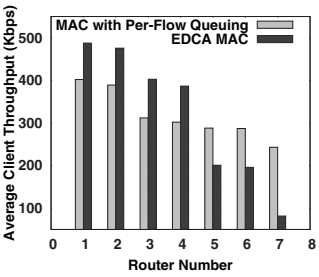


Fig. 4. Clients' avg TCP throughput per router

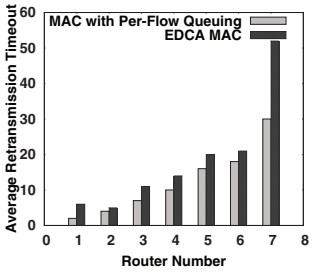


Fig. 5. TCP unfairness analysis

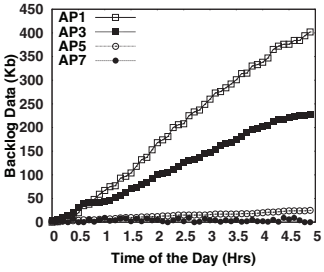


Fig. 6. Router backlog data

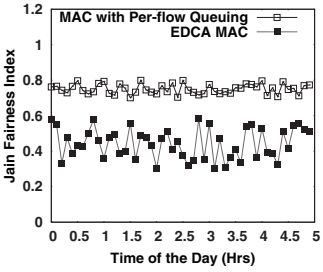


Fig. 7. Jain fairness index

B. Experimental Results and Analysis

Fig. 2 shows the average client throughput for every router in the testbed. The average throughput for clients under mesh routers near the gateway is maximum, and throughput degrades

as the number of hops increases. The degradation in UDP throughput is much higher for the standard EDCA MAC compared to the fair-queuing approach. However even in fair queuing, the average UDP throughput for clients under AP7 is almost 50% of the average UDP throughput for clients under AP1. This indicate that unfairness still exist among clients in different hop-distances, and the unfairness becomes more prominent as number of hops increases. Fig. 3 shows the packet delivery rate of individual routers, which indicates that all mesh routers remain in saturation state. Though fair-queuing approach controls the rate of incoming traffic from upper layers, it is independent of the MAC layer traffic forwarding. As an example, clients under AP5 and AP6 are unaware of the forwarding load from AP7. Once AP5 and AP6 get saturated, clients under AP7 starts suffering. As a result, though the data delivery rate for AP7 is very high for EDCA MAC, the average client throughput is very low. Most of the packets transmitted by AP7 are lost at next-hop APs (AP5 and AP7) due to the buffer overflow from the interface queues.

Fig. 4 shows the average client throughput for TCP traffic. TCP has in-built fairness assurance property through the window based flow-control mechanism, that keeps the average bandwidth usage of every TCP flow almost equal. The results obtained from our experiments using TCP traffic revealed that fairness is assured only among clients of the same router. There is severe unfairness among clients from different routers. Fig. 5 explains the reason by explaining TCP behavior using TCP retransmission timeout. TCP retransmission timeout occurs if the TCP client does not receive the acknowledgement packet within a timeout interval (generally kept slightly larger than the round trip time). If a retransmission timeout occurs, TCP decreases the window size to trigger flow control activities. From figure 5, it can be observed that the average retransmission timeout increases exponentially as the number of hops increases. This indicate that the TCP congestion detection and flow control actions are triggered more number of times, as the traffic has to traverse more number of hops. Fig. 6 shows the backlog data of four routers collected over a period of 5 hours. The backlog data at AP1 (which is more closer to the gate) increases rapidly compared to AP3, and the trend follows. A frame generated at the client of AP7 has to wait at the interface queues of AP5, AP3 and AP1, before it gets delivered to the gateway. The waiting time at the queues increases with the number of hops, and after few hops, the waiting time becomes sufficient to affect the retransmission timer.

The third experiment with mixed TCP and UDP applications has been conducted over a period of 5 hours to get an overview of the fairness statistics. Jain Fairness Index [13] is used as the quantitative measure of the fairness. The fairness index value 1 indicates perfect fairness. Fig. 7 plots the result with respect to the time. EDCA MAC performs very poorly while fair queuing improves fairness among the users. Instead the fairness index is never more than 0.8 even with fair-queuing. This indicates that at least 20% of clients suffer from unfairness in the whole network. Further, sometime the index value even drops near to 0.7. The analysis has confirmed that the index value drops significantly when the traffic load increases suddenly in the network (it has been observed that some file transfer activities have been initiated at that time).

The experiments show that the MAC layer coordination is necessary to ensure fairness in a mesh network, such that the backlog data at mesh routers can be minimized for the perfect operation of the transport layer fairness control. The results reveal that the end-to-end flow control mechanism can not perform well because of the excessive packet losses due to the overflow from the MAC layer interface queue. Therefore, a coordination is required among the MAC layer and the transport layer flow control mechanisms. Though TCP handles buffer overflow by reducing the window size, it helps to control congestion, and to maintain fairness among clients under the same router. However, severe unfairness still exist among clients under different routers at different hop distance away. The unfairness becomes worse with high traffic load, due to the increasing contention among mesh routers, as well as packets overflows from router buffers.

III. MAC Layer Proportional Fairness

The previous protocol proposed in [8] ensures the MAC layer proportional fairness (channel share is proportional to the traffic demand) to provide equal bandwidth share among every client, in spite of their hop-distance from the gateway. This section provides improvements and optimization over the proportional fairness protocol to support it in real testbed.

A. Background of the Fairness Protocol

The proportional fairness protocol proposed in [8] works as follows,

(i) Every mesh router estimates the total traffic load using the traffic demand for the up-link and the down-link clients. The total load of a router is termed as the *'Activity Factor'* (AF).

(ii) Every router estimates the required channel share as the ratio of its own AF value to the summation of AF values of all its neighbors (more specifically, two hop neighborhood that can contend for channel access).

(iii) The routers further estimate the *actual channel share* from the previous history. This is calculated as the ratio of number of packets transmitted by this routers to the number of overheard packets (packets from neighbors).

(iv) By comparing the required channel share with the actual channel share, every router enters one of the states - *restrictive* (actual channel share is more), *aggressive* (actual channel share is less) and *normal*.

(v) Based on the current state, mesh routers tune their CW values to prioritize or to deprive their channel share. In aggressive mode, CW value is decreased to prioritize channel share, and in restrictive mode, CW is increased to deprive channel share.

The protocol has two fundamental shortcomings. First, the total load is estimated using traffic demand of individual clients, that is a time varying metric, and second, the CW

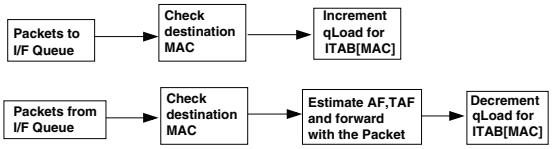


Fig. 8. Estimation of AF value

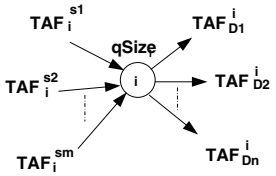


Fig. 9. Load Propagation

tuning mechanism may result in instability when new flows join, and existing flows are terminated. To assure equity among the end users by providing proportional fairness at the MAC layer, following two conditions must be satisfied,

C.1. The backlog data at the interface queue of all mesh routers should be independent of hop-distance from the gateway. Ideally, the backlog data should be zero at all mesh routers to assure equal waiting time for all flows. This assures that the transport layer flow control mechanism would be able to deliver data based on the end-to-end path statistics, and all the flows would be treated in the similar way, in spite of their hop distance from the gateway. Reducing the backlog data at the interface queue assures that the local flows (flows originated from clients under a router) do not reserve the full capacity of the router.

C.2. The transport layer flow control protocol should consider the effective delivery rate of the router for the flow assignment such that the backlog data at the interface queue is minimized.

If the above two conditions are satisfied, the waiting times of the packets for every flow at the interface queue become negligible, and the transport layer flow control protocol works perfectly. To assure condition C.1, estimation of AF value at every mesh router should consider the backlog data at the interface queue, rather than the traffic demand of individual clients. This way AF value can be computed locally at every router without the possibility of the stale information.

B. Estimation of AF Value

The AF estimation procedure is shown in Fig. 8. Every router stores a table at MAC layer, termed as $ITAB$ that contains the MAC addresses of all its one-hop neighbors and a corresponding entry, called $qLoad$. Whenever a packet is enqueued to the interface queue, the $qLoad$ entry in the $ITAB$ for corresponding destination MAC is incremented by one. Similarly, whenever a packet is dequeued from the interface queue, the corresponding $qLoad$ entry in the $ITAB$ for destination MAC is decremented. Before forwarding the packet, every router estimates the AF value and a portion of the AF value, called the *transferred AF* (TAF), is piggybacked with the packet. The AF and TAF values are estimated as follows.

Let AF_i denotes the AF value for router i. AF_i is calculated as,

$$AF_i = \sum_{k \in \mathbb{N}_i} TAF_i^k + qSize_i \quad (1)$$

where TAF_i^k denotes TAF from router k to router i, $qSize_i$ is the size of the interface queue of router i and $\mathbb{N}_i$ is the set of one-hop neighbors. If a router $k \in \mathbb{N}_i$ does not transfer data to router i, then $TAF_i^k = 0$. Let $qLoad_k$ denotes the entry in $ITAB$ for the MAC address of the destination router k. Then TAF_i^k can be represented as,

$$TAF_i^k = AF_k \times \frac{qLoad_i}{qSize_k} \quad (2)$$

Let us consider Fig. 9. TAF represents the amount of traffic to be relayed from the neighbors and $qSize_i$ represents amount of backlog traffic (both the relayed and from the client). Therefore, router i should get sufficient channel access to forward the complete traffic to clear its interface queue. Further, the status of the interface queue gives an estimation of the percentage of data to be forwarded to a specific neighbor for multiple forwarders. It can be noted that when the backlog data is very small, the system is already in fair state. The problem occurs when the traffic load is high, resulting in large amount of backlog data at the interface queues. It can be shown theoretically through queuing analysis that equation (1) and equation (2) provide a good estimation when traffic load is high. However, the analysis is not given here due to space constraint. Based on the estimation of AF value, the *required channel share* of every router can be calculated using the similar procedure proposed in [8].

C. Cross Layer Interaction

As discussed earlier, condition C.2 is required to control the incoming traffic from the transport layer using efficient flow control mechanism. It has been observed that almost all the existing flow control algorithms consider the actual data rate of a mesh router for the flow assignment. There exist very few approaches, such as back-pressure routing [14] and its variants, where mesh routers near the gateway falsely create a congestion effect to reduce the data rates for the flows originated from away routers. However, back-pressure effect introduces extra delay in the network [15]. In the current approach, a cross layer interaction mechanism is used. Let *required channel share* and actual physical data rate for router i be RS_i and λ. Then the effective delivery rate for router i, denoted as Z_i, can be expressed as,

$$Z_i = RS_i \times \lambda \quad (3)$$

This information is periodically forwarded to the transport layer flow control protocol. The flow control mechanism assigns flows based on the effective delivery rate, and condition C.2 can be assured.

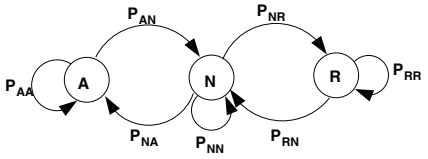


Fig. 10. Three state Markov model depicting state transition

D. CW Tuning based on Statistical Analysis

The CW tuning mechanism proposed in [8] considers only the short term effect, and thus may trap into local performance degradation, when new flows are introduced or existing flows terminate. This is because, sudden changes in the traffic load are not considered in the CW tuning mechanism. For example, let a new file transfer application be introduced in router i. Then the traffic for router i increases suddenly. However, the overall condition of the network cannot be stimulated until the new flow is completely distributed in the network, and the network becomes steady. Therefore, without predicting the future network stability condition, router i may suddenly increase its channel demand, that may affect its neighbors.

As discussed earlier, every router maintains three states - *aggressive*, *normal* and *restrictive*. Based on the current state, mesh routers either decrease or increase their CW values to either prioritize or to deprive channel access. In the proposed protocol, mesh routers do not change the state immediately, if they find differences in the required channel share and the actual channel share. Rather, they do a statistical analysis based on a three-state discrete time Markov model to check whether state change is required or not. Fig. 10 shows the three state Markov model for the state transition, where state 'A' represents the *aggressive* mode, state 'N' represents the *normal* mode and state 'R' represents the *restrictive* mode. The transition probabilities P_{uv} depicts the probability of transition from state u to state v, where $u, v \in \{A, N, R\}$. Every router maintains the time duration it is in the current state, and it was in the immediate past state. Let t_A, t_N and t_R denotes the time duration the router is in state 'A', state 'N' and state 'R'. Based on the completeness of Markovian process, the transition probabilities are represented as follows;

$$P_{AN} = \frac{t_A}{t_A + t_N}; \qquad P_{AA} = 1 - P_{AN}; \quad (4)$$

$$P_{RN} = \frac{t_R}{t_N + t_R}; \qquad P_{RR} = 1 - P_{RN}; \quad (5)$$

$$P_{NA} = \frac{t_N}{t_A + t_N + t_R}; \quad (6)$$

$$P_{NR} = \frac{t_N}{t_A + t_N + t_R}; \quad P_{NN} = 1 - P_{NA} - P_{NR}; \quad (7)$$

Assume router i wants to transit from state u to state v. Let AF_{iu} denote the average AF value till now, when the router is in state u, and AF_{ic} is the current AF value for which the router wants to change its state based on the comparison between the actual channel share and the required channel share. The transition decision is respected if and only if $P_{uv} \times AF_{ic} > P_{uu} \times AF_{iu}$, otherwise the router does not changes its state.

In the proposed protocol, the decision of changing state is based on the statistics of AF values and the corresponding state transition probability. It depicts whether the changes in the AF values are sufficient enough to change the current state. Thus small changes in the AF values do not trigger state changes, if the current state is stable enough. This statistical decision reduces unnecessary state transitions, and avoids trapping in the local performance degradation.

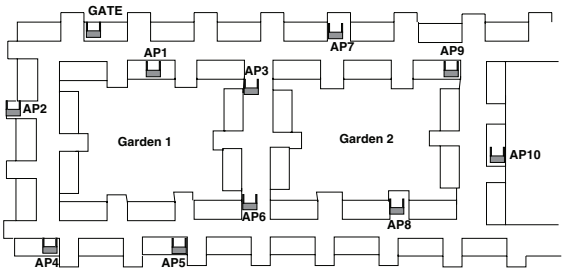


Fig. 11. Location of nodes in the 11 node testbed

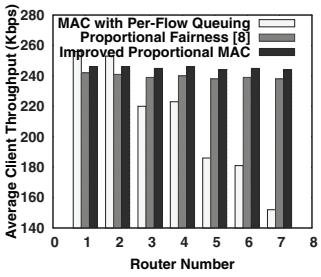


Fig. 12. Throughput/Client (N8)

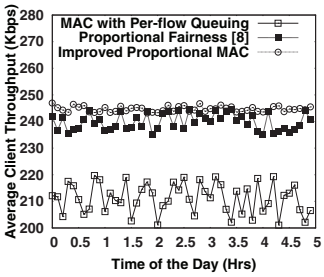


Fig. 13. Average Throughput (N8)

IV. Experimental Results

The proposed protocol has been implemented using Ralink SDK as an extension to the existing 802.11 EDCA MAC. Two different topologies have been used for the experiments. The first topology is the earlier one, as shown in Fig. 1. Another sparse topology has been used, with 11 APs, as shown in Fig. 11. Though the second topology can sustain only at $2Mbps$ data rate using IEEE 802.11b, the worse case analysis is possible with maximum network load with different types of application traffic. All the three batches of experiments, as stated in Section 2, have been conducted. However, the results for only the third experiment (with simultaneous different types of application) have been reported and analyzed here for space constraints. In the figures, 'N8' denotes the 8 node testbed, and 'N11' denotes the 11 node testbed.

Fig. 12 shows the average clients' throughput per router from the 8 node testbed. It can be noted that every router operates at $6Mbps$. The proportional fairness protocol proposed in [8] improves fairness among clients by distributing the channel bandwidth proportionally among mesh routers, and evenly among clients. The improved load estimation mechanism proposed in this paper (mentioned as the 'improved proportional MAC' in the graphs) further improves the average

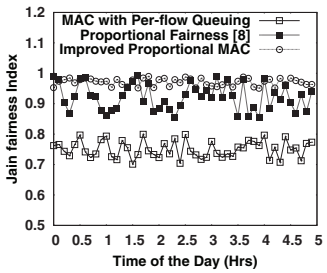


Fig. 14. Jain Fairness Index (N8)

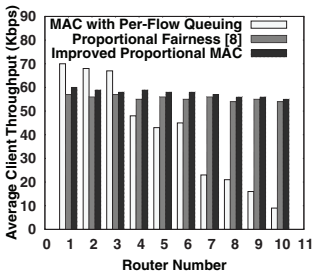


Fig. 15. Throughput/Client (N11)

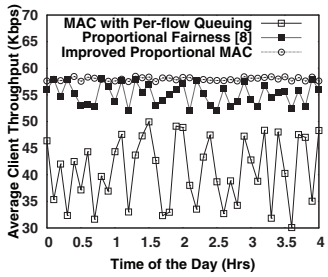


Fig. 16. Average Throughput (N11)

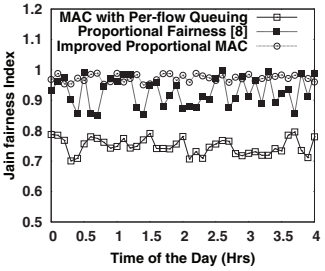


Fig. 17. Jain Fairness Index (N11)

throughput, as well as the fairness, by avoiding the local traps during the state change of a mesh router. This is visible clearly from Fig. 13 that shows the average throughput of all clients with respect to the time. Though the average clients' throughput for the protocol proposed in [8] is more than the EDCA MAC with fair-queuing, the throughput value drops several times due to the local trapping as discussed earlier. The proposed protocol in this paper maintains constant throughput with respect to the time. The average throughput is improved around 14% for the protocol proposed in [8], and around 16.7% for the proposed protocol in this paper, compared to the EDCA with fair-queuing. The comparison in Jain fairness index is shown in Fig. 14. For the proposed protocol, a high value of the fairness index is maintained constantly with respect to the time.

Similar trend is followed for the 11 node testbed, as shown in Fig. 15 to Fig. 17. However, average throughput is less compared to the 8 node testbed, as mesh routers can sustain maximum at $2Mbps$ rate compared to $6Mbps$ in the 8 node testbed. For such an overloaded network, the percentage improvement in average throughput is considerably high, as shown in Fig. 16. In the 11 node testbed, the average throughput is improved around 35.37% for the protocol proposed in [8], and around 40.49% for the proposed protocol in this paper, compared to the EDCA with fair-queuing. Fig. 17 shows the fairness index with respect to the time. The index value drops several times in case of the earlier protocol [8], because of the false CW tuning as a result of the stale information from clients. In the protocol presented in this paper, the average fairness index maintains a steady value at 0.95 in average. This shows that even for a network with heavy traffic load, the proposed protocol is efficient to provide equity among end-users, and improves average network throughput considerably.

V. Conclusion

This paper studies the fundamental reason of unfairness in IEEE 802.11 wireless mesh networks using results obtained from a practical testbed. Clients associated with the routers near the gateway reserve the full capacity, as a consequence of which, the flows with large number of hops get starved. The analysis shows that MAC layer proportional fairness is required for correct performance of transport layer end-to-end fair scheduling algorithms. This paper proposes an improved load estimation and CW tuning strategy over the previously proposed proportional fairness protocol. The effectiveness of the proposed protocol is justified through experimental results obtained from two different testbed setups.

VI. Acknowledgement

The work of the first author is supported by TATA Consultancy Services Research Fellowship Program, India. We would like to thank Mr. Rajendra Singh, Skiva Technologies for providing necessary hardware and technical supports.

References

[1] D. Benyamina, A. Hafid, and M. Gendreau, "Wireless mesh networks design - a survey," *IEEE Communications Surveys & Tutorials*, vol. 14, no. 2, pp. 299–310, 2012.

[2] I. Ahmed, A. Mohammed, and H. Alnuweiri, "On the fairness of resource allocation in wireless mesh networks: a survey," *Wireless Networks*, pp. 1–18, 2013.

[3] V. Gambiroza, B. Sadeghi, and E. W. Knightly, "End-to-end performance and fairness in multihop wireless backhaul networks," in *Proc. of the 10th MobiCom*, 2004, pp. 287–301.

[4] A. Eryilmaz and R. Srikant, "Joint congestion control, routing, and mac for stability and fairness in wireless networks," *IEEE Journal on Selected Areas in Communications*, vol. 24, no. 8, pp. 1514–1524, 2006.

[5] Y.-S. Chen, F.-C. Tseng, and C.-H. Ke, "Achieving weighted fairness for high performance distributed coordination function with QoS support in WLANs," *Advances in Intelligent Systems and Applications*, pp. 625–634, 2013.

[6] J. Jeong, J. Choi, S. Choi, and C.-k. Kim, "Resolving intra-class unfairness in 802.11 EDCA," *Wireless Personal Communications*, vol. 63, no. 2, pp. 431–445, 2012.

[7] S. Pudasaini, S. Shin, and K. Kim, "Carrier sense multiple access with improvised collision avoidance and short-term fairness," *Wireless Networks*, vol. 18, no. 8, pp. 915–927, 2012.

[8] S. Chakraborty, P. Swain, and S. Nandi, "Proportional fairness in MAC layer channel access of IEEE 802.11s EDCA based wireless mesh networks," *Ad Hoc Networks*, vol. 11, no. 1, pp. 570 – 584, 2013.

[9] Ralink RT-3352 router-on-chip. [Online]. Available: http://www.mediatek.com/_en/01_products/04_pro.php?sn=1006

[10] M. Al-Rabayah and R. Malaney, "A new hybrid location-based ad hoc routing protocol," in *Proc. of the IEEE Globecom*, 2010, pp. 1–6.

[11] J. Jun and M. L. Sichitiu, "Fairness and QoS in multihop wireless networks," in *Proc. of IEEE 58th VTC*, vol. 5, 2003, pp. 2936–2940.

[12] T. Bonald and L. Massoulié, "Impact of fairness on Internet performance," *ACM SIGMETRICS Performance Evaluation Review*, vol. 29, no. 1, pp. 82–91, 2001.

[13] R. Jain, D. M. Chiu, and W. Hawe, "A quantitative measure of fairness and discrimination for resource allocation in shared computer systems," Digital Equipment Corporation, DEC-TR-301, Tech. Rep., Sep. 1984.

[14] S. Moeller, A. Sridharan, B. Krishnamachari, and O. Gnawali, "Routing without routes: The backpressure collection protocol," in *Proc. of the 9th ACM/IEEE IPSN*, 2010, pp. 279–290.

[15] A. Dvir and A. V. Vasilakos, "Backpressure-based routing protocol for DTNs," *ACM SIGCOMM Computer Communication Review*, vol. 40, no. 4, pp. 405–406, 2010.

A Trigger Counting Mechanism for Ring Topology

Sushanta Karmakar[1] **Subhrendu Chattopadhyay** [1]

[1] Department of Computer Science and Engineering,
Indian Institute of Technology Guwahati, India, 781039
Email: sushantak@iitg.ernet.in, subhrendu@iitg.ernet.in

.

Abstract

Consider a distributed system with n processors, which receive triggers from the outside world. The Distributed Trigger Counting (DTC) problem is to raise an alarm if the number of triggers over the system reaches w, which is an user specified input. DTC is used as a primitive operation in many applications, such as distributed monitoring, global snapshot etc. In this paper, we propose an algorithm for the DTC problem in a ring topology with a message complexity of $O(n^2 \log(w/n))$ and each node in the system receives $O(n \log(w/n))$ number of messages. We also discuss about the possible tuning of the algorithm which results better complexities.

Keywords: Distributed algorithm, distributed monitoring, distributed trigger counting, ring topology

1 Introduction

Distributed trigger counting (DTC) is an important problem in distributed systems. Consider a distributed system with n processes. Each process receives some triggers (signals) from an external source. The DTC problem is to detect the state when the number of triggers received by the system reaches w which is a user defined input to the system. Note here that w may be much larger than n. The sequence of processors receiving the w triggers is not known apriori to the distributed system. Our goal is to propose an algorithm for the DTC problem under a known topological setting. More specifically, in this paper we propose an algorithm for the DTC problem in a ring network.

The DTC problem arises in many applications in distributed systems. Some major application areas can be distributed monitoring, global snapshot etc. Monitoring is an important aspect in wireless networks as well as in wired networks. A wireless sensor network is typically used to monitor physical or environmental conditions such as border surveillance, forest fire detection, traffic management in highways etc. In traffic management, one may be interested in detecting whether the number of vehicles on a highway exceeds a certain threshold. Similar applications may be found in border surveillance and forest fire detection as well. In distributed system a global snapshot state is said to be valid provided all the in-transit messages are recorded. It was shown by Garg et al. (2006) that the problem of detecting whether all the in-transit messages have been recorded can be reduced to the DTC problem. Awerbuch (1985) proposed the concept of synchronizers which is a tool to transform a synchronous distributed algorithm to another that runs on an asynchronous distributed system. Here a distributed system is required to generate a signal (or pulse) when all the messages generated after the previous signal have been delivered. This problem can also be formulated as a variant of the DTC problem. The performance of an algorithm for the DTC problem is often measured by the following two metrics.

- *Message complexity:* It is the total number of messages sent by all the nodes of the systems.
- *MaxRcvLoad:* It is the maximum number of messages received by any node in the system.

The DTC problem was studied by Garg et al. (2006) for a general distributed system. In their work, they proposed two algorithms for the DTC problem: a centralized algorithm and a tree-based algorithm. The centralized algorithm has a message complexity of $O(n \log w)$. Its MaxRcvLoad has a complexity of $O(n \log w)$. This is a tight bound for the centralized algorithm. The tree based algorithm has a message complexity of $O(n \log n \log w)$. Again the MaxRcvLoad of the tree based algorithm has a complexity of $O(n \log n \log w)$. In the work the authors also showed that any deterministic algorithm for the DTC problem must have a message complexity of $\Omega(n \log(w/n))$. Huang et al. (2007) proposed a novel solution to the problem of efficient detection of an aggregate predicate over cumulative triggers over a time-varying window in a distributed monitoring system. They provided a queueing theory based analysis of the problem which provides the user the power to trade-off between desired accuracy and communication overhead. However, their approach is based on a single coordinator. Hence the algorithm is not fully distributed. In another work (Chakaravarthy, Choudhury, Garg & Sabharwal 2011) a randomized distributed algorithm was proposed for trigger counting in a tree based topology. The algorithm has a message complexity of $O(n \log n \log w)$ and MaxRcvLoad complexity of $O(\log n \log w)$ with high probability. Later in another work (Chakaravarthy, Choudhury & Sabharwal 2011) the authors proposed an improved algorithm for the DTC problem having message complexity of $O(n \log w)$ and MaxRcvLoad of $O(\log w)$. However, this work has the limitation that the algorithm is a randomized algorithm and it does not provide any bound on the messages sent per node.

Table 1: Summary of DTC algorithms

Algorithm	*Messages*	*MaxRcvLoad*	*Nature*
Tree-based (Garg et al. 2006)	$O(n \log n \log w)$	$O(n \log n \log w)$	randomized
Centralized (Garg et al. 2006)	$O(n \log w)$	$O(n \log w)$	centralized
(Chakaravarthy, Choudhury, Garg & Sabharwal 2011)	$O(n \log n \log w)$	$O(\log n \log w)$	randomized
(Chakaravarthy, Choudhury & Sabharwal 2011)	$O(n \log w)$	$O(\log w)$	randomized
(Emek & Korman 2010)	$O(n(\log n \log w)^2)$	$O((\log n \log w)^2)$	deterministic
Unidirectional Ring (this paper)	$O(n^2 \log \frac{w}{n})$	$O(n \log \frac{w}{n})$	deterministic

Also this algorithm assumes a clique of n nodes as the topology. In fact they concluded that designing a deterministic algorithm with message complexity of $O(n \log w)$ and *MaxRcvLoad* of $O(\log w)$ for any arbitrary network is an open DTC problem.

In this paper, we propose a deterministic distributed algorithm for the DTC problem in a unidirectional ring network. A ring of nodes is an interesting topology and is used in solving many distributed computing problems such as leader election, mutual exclusion, distributed snapshot etc. Also if a solution of a problem in distributed computing can be obtained for a ring topology then the same can be applied to have a solution for the same problem under arbitrary topology by means of embedding a virtual ring topology over the network (e.g. Eulerian graph). Therefore for many reasons a ring topology is important and in this paper we propose a solution to the DTC problem assuming a ring topology. We also provide an outline of how the DTC problem can be solved in a general network using our proposed solution to the problem in a ring network.

The main challenge of the DTC problem is that outside triggers may be detected by any node and one would like to minimize the communication overhead for determining when a total of w external triggers have been counted. It is important to see that there is a tradeoff between minimizing the communication overhead and having a timely detection of the w triggers. The simplest way to minimize the number of messages is to let every node detect triggers and when any one has detected w triggers, the global alarm signal is sent. However, in this simple scheme there is obviously a big risk that the system will be seriously delayed in sending the global alarm signal which has to be sent as soon as w or more triggers are received by all the nodes. The obvious way to avoid such delays is to send a message as soon as a trigger is detected. However this will generate a lot of messages (potentially $O(w)$). Hence there should be some compromise between these two extremes. In an earlier work (Garg et al. 2006) it was shown that any deterministic algorithm for the DTC problem must have a message complexity lower bound of $\Omega(n \log(w/n))$. However this result was analytical and no algorithmic instance has been found so far satisfying this lower bound.

Let there be n nodes in a unidirectional ring. The nodes are denoted as v_1, v_2, v_3, ... , v_n. Any node v_i can send a message to v_{i+1} and receive a message from v_{i-1}. Specifically, v_n sends a message to v_1. We assume an asynchronous model of computation and communication (via messages). We assume that the channels are reliable and FIFO. There is no node or link failure. Also messages are not corrupted or spuriously introduced. It is assumed that each node has a unique identifier. Out of the n nodes, there is one node v_n designated as the *master*. All other nodes (v_i such that $i \neq n$) act as slaves. The algorithm is stated in the form of guarded statements. A guarded statement is of the form $g \rightarrow a$ where g is the guard, and a is the action. The action a is executed if and only if g is true. The program for any node i contains a sequence of statements $\{S_1, S_2, \ldots, S_n\}$ where each S_j is of the form $g_j \rightarrow a_j$. The j-th statement of the program at node i is denoted by $S_j(i)$. The guard corresponding to $S_j(i)$ is denoted by $g_j(i)$ and the action corresponding to $S_j(i)$ is denoted by $a_j(i)$. Also it is assumed that the guarded actions are atomic. The main result of our algorithm is as follows. The distributed trigger counting algorithm over a ring of n processors has a message complexity of $O(n^2 \log \frac{w}{n})$ and *MaxRcvLoad* of $O(n \log \frac{w}{n})$ where w is the number of triggers to be counted.

The rest of this paper is organized as follows. Section 2 contains the related work on the DTC problem. The proposed algorithm for the DTC problem in a unidirectional ring is presented in Section 3. Section 4 contains the analysis for proving the correctness and message complexity of the proposed algorithm. Section 5 discusses the tuning of the algorithm with respect to some parameters. The trade-off between message complexity and delay in raising the alarm is discussed in Section 6. Solving the DTC problem under arbitrary network is discussed in Section 7. Finally we conclude in Section 8.

2 Related Work

Many of the earlier works primarily consider the DTC problem in a centralized setting and solve it using randomized algorithm. A randomized algorithm was proposed by Emek & Korman (2010) for the DTC problem with message complexity of $O(n(\log \log n)^2 \log w)$ and average message complexity of $O((\log \log n)^2 \log w)$. In this paper it is assumed that the input is constructed by an adaptive adversary whose decisions may depend on previous coin tosses of the randomized protocol but not on future ones. Here a separator decomposition of a tree of n nodes is constructed over which the events are aggregated. By changing the internal parameters of the randomized protocol they have obtained a deterministic protocol with message complexity of $O(n(\log n \log w)^2)$ and MaxRcvLoad of $O((\log n \log w)^2)$.

A fundamental class of problems called "thresholded counts" was introduced by Keralapura et al. (2006) where the goal is to return the aggregate frequency count of an event, that is continuously monitored by distributed nodes with a user-specified accuracy, whenever the actual count exceeds a given threshold value. They studied the cases under static as well as dynamic threshold values. Cormode et al. (2011) defined a function monitoring problem as a 4-tuple $(\kappa, f, \tau, \epsilon)$ where κ denote the number of nodes, f denote the function that is being monitored by the nodes, τ denote the threshold such that if $f \geq \tau$ then the system generates some alarm 1 and the alarm is 0 if $f \leq (1-\epsilon)\tau$. The authors give lower and upper bounds of communication cost for the $(\kappa, f, \tau, \epsilon)$

MasterProcess(*TriggerCount* w) // for master node v_n

Initialization: *TargetTriggerCount* $w' = w$; $C = 0$; $sflag = true$; $fsorFlag = false$; $eorFlag = false$; ψ; τ

(S_1) $sflag \rightarrow \psi = w'/2$; $\tau = w'/2n$
send $< start - of - round, \tau >$ to v_1
$fsorFlag = true$; $sflag = false$

(S_2) upon receiving $< Trigger > \wedge fsorFlag \rightarrow C = C + 1$

(S_3) upon receiving $< Coin, factor > \wedge fsorFlag \rightarrow C = C + factor \times \tau$

(S_4) $fsorFlag \wedge (C \geq \psi) \rightarrow efactor = 0$
Send $< end - of - round, efactor >$ to v_1

(S_5) upon receiving $< end - of - round, efac >$ from $v_{n-1} \rightarrow$
$w' = w' - (C + efac \times \tau)$
if $(w' > 0)$ then $sflag = true$
else $sflag = false$
$fsorFlag = false$
send $< TargetReached >$ to v_1

Figure 1: Pseudocode of master node for DTC algorithm for a ring

problem with different f. The algorithms proposed by the authors are randomized. One example of f can be the aggregate function. Here each input trigger i is associated with a value α_i. The goal is to raise an alarm when the aggregate of these values crosses a threshold.

A decentralized and randomized algorithm called LayeredRand algorithm was proposed by Chakaravarthy, Choudhury, Garg & Sabharwal (2011). In this work, the nodes are organized in a tree topology and they communicate only with the nodes in the adjacent layers. The algorithm proceeds in multiple rounds and in each round it achieves a trigger count which is approximately half of the required count. In another work, Chakaravarthy, Choudhury & Sabharwal (2011) proposed an approximate algorithm with the assumption $w \leq 2^n$, and it exhibits a message complexity of $O(n \log n \log w)$ and MaxRcvLoad of $O(\log n \log w)$. Similar works were done in (Korman & Kutten 2007, Emek & Korman 2011). No prior work on the DTC problem considered a ring as the underlying topology.

3 Algorithm for DTC in a Unidirectional Ring

In this section, we describe an algorithm for the DTC problem in a unidirectional ring topology. Its message complexity is $O(n^2 \log(w/n))$ and *MaxRcvLoad* is $O(n \log(w/n))$. We assume that the triggers which come from the outside world are distributed to the n nodes of the ring randomly. The distribution of the triggers among n nodes is arbitrary and not known apriori. The algorithm consists of a number of rounds. The total number of triggers to be counted by the system is denoted by w. The system has n number of nodes connected in a logical unidirectional ring configuration. The nodes in the ring are denoted as v_i $(1 \leq i \leq n)$. There is one node v_n which is designated as the *master* node. All the other nodes in the ring are called *slaves*. Each *slave* node has the following state variables: C is a counter indicating the number of received triggers that have not yet contributed to the distributed trigger counting and initialized to 0; $fsorFlag$ is a boolean flag which indicates the start of the DTC algorithm, and it is initialized to $false$; $cflag$ is a boolean flag which indicates whether a node has received a *Coin* message, and it is initialized to $false$; *coinCount* is an integer that denotes the number of *Coin* messages sent by the node; $eorFlag$ is a boolean flag that indicates the end of the current round and it is set to $true$ on receipt of a *end-of-round* message. For the *master*, there are some other state variables in addition to the aforesaid variables. They are: w' is the remaining number of triggers yet to be counted by the system and is initialized to w; $sflag$ is the start of round flag for each round and it is initialized to $true$. The *master* node also maintains two variables, ψ and τ, which are known as *global threshold* and *local threshold*. Each slave node gets the value of τ for the current round from its previous node in the ring. To reduce the number of messages, each slave node sends a *Coin* message only when its trigger count crosses the local threshold, τ. Similarly the *master* node sends a *end-of-round* message only when the total trigger count at the master crosses its global threshold, ψ.

For any node v_i, v_{i-1} is called the predecessor of v_i and v_{i+1} is the successor of v_i. Note that the successor of v_n is v_1. Similarly the predecessor of v_1 is v_n. The trigger counting algorithm proceeds in multiple rounds. At the start of each round, the system should know the number of triggers which are yet to be counted to raise an alarm. This can be known by subtracting the sum of all the triggers received in the previous round from w'. Each round of the algorithm is started by the *master* node v_n by sending the *start-of-round* message to v_1. Node v_n calculates ψ and τ, and sends τ to v_1 along with the *start-of-round* message. It updates its state variables as follows: $fsorFlag(v_n) = true$ and $sflag(v_n) = false$. When a *slave* v_i gets (*start-of-round*, τ') message it also calculates its local threshold as $\tau = \tau'$. It sets its local state variables as follows: $fsorFlag(v_i) = true$, $eorFlag(v_i) = false$. Since it has not yet sent any *Coin* message, v_i sets $coinCount(v_i) = 0$. It also sends a (*start-of-round*, τ) message to v_{i+1}.

With $fsorFlag(v_i) = true$ $(1 \leq i \leq n)$, any node v_i increments its local counter $C(v_i)$ on receiving a *Trigger* message. So each node v_i independently

SlaveProcess() // code for node v_i where $i \neq n$

Initialization: $C = 0$; $fsorFlag = false$; $cflag = false$; τ; $coinCount = 0$; $factor$; $eorFlag = false$

(S_6) upon receiving $< start - of - round, \tau' > \rightarrow$ $\tau = \tau'$
Send $< start - of - round, \tau >$ to v_{i+1}
$coinCount = 0$
$fsorFlag = true$; $eorFlag = false$

(S_7) upon receiving $< Trigger > \wedge\ fsorFlag \rightarrow C = C + 1$

(S_8) upon receiving $< Coin, fac > \wedge\ fsorFlag \wedge \neg cflag \rightarrow C = C + fac \times \tau$
$cflag = true$

(S_9) $fsorFlag \wedge (C \geq \tau \vee cflag) \wedge \neg eorFlag \wedge coinCount \leq n \rightarrow$ $factor = C/\tau$
Send $< Coin, factor >$ to v_{i+1}
$C = C - factor \times \tau$
$coinCount = coinCount + 1$
$cflag = false$

(S_{10}) upon receiving $< end - of - round, efactor > \rightarrow$
$eorFlag = true$
$factor = C/\tau$
$C = C - factor \times \tau$
$efactor = efactor + factor$
Send $< end - of - round, efactor >$ to v_{i+1}

(S_{11}) upon receiving $< TargetReached > \rightarrow$ $fsorFlag = false$
send $< TargetReached >$ to v_{i+1}

Figure 2: Pseudocode of slave node for DTC algorithm for a ring

counts the number of triggers it receives. Whenever v_i finds that $C(v_i) \geq \tau$ or $cflag(v_i) = true$ then it sends a *Coin* message to v_{i+1} $(i \neq n)$. Note here that $cflag(v_i) = true$ only if v_i has received a *Coin* message from its previous node in the ring. In this case v_i computes the $factor = C/\tau$ and sends the $factor$ along with the *Coin* message to its successor node v_{i+1} in the ring. So a *Coin* message essentially transfers $\tau \times factor$ amount of triggers from v_i to the successor v_{i+1}. Node v_i also reduces the value of its counter C appropriately and remembers the send of a *Coin* message by incrementing the variable *coinCount*. Then it sets $cflag(v_i) = false$. It is clear from S_9 in the pseudocode of Figure 2 that a slave can send at most n number of *Coin* messages in a round of the algorithm.

If a node v_i $(i \neq 1)$ receives a *Coin* message from its predecessor and $fsorFlag(v_i) = true$ and $cflag(v_i) = false$ then v_i increments it local trigger count $C(v_i)$ by $\tau \times fac$, where fac is the multiplicative factor received along with the *Coin* message. Also v_i sets $cflag(v_i) = true$. Therefore *Coin* messages move from one node to another and eventually reaches the *master* node v_n. When the *master* receives a *Coin* message, v_n increments $C(v_n)$ by $\tau \times fac$, where fac is the the multiplicative factor that v_n received from its predecessor. When the *master* finds that $C(v_n) \geq \psi$ then v_n initiates the end of the round by sending a *end-of-round* message to v_1. During the propagation of the *end-of-round* message from one node to another, nodes continue to receive triggers from the outside world. These triggers are counted in a similar way.

If a slave v_i receives (*end-of-round*, $efactor$) message then it first set $eorFlag(v_i) = true$ such that no further *Coin* message is sent in this round. It computes $factor = C/\tau$ and adds this with the received multiplicative factor $efactor$, and sends this modified $efactor$ to v_{i+1} along with a *end-of-round* message. In this way the (*end-of-round*, $efactor$) message moves from one slave to another and eventually reaches the *master*. When the *master* receives (*end-of-round*, $efac$) message from v_{n-1} it computes the remaining number of triggers yet to be counted in future rounds. This is given by $w' = w' - (C(v_n) + efac \times \tau)$. If the remaining number of triggers yet to be counted is more than zero then v_n sets it variable $sflag(v_n) = true$ so that it can start another round (by S_1 of Figure 1).

4 Correctness and Analysis

Lemma 1. *A slave node sends at most n number of* Coin *message in each round.*

Proof. By S_9, a node v_i sends a *Coin* message to v_{i+1} if the G_9 is true. Again G_9 is true if $coinCount \leq n$. By S_6, $coinCount = 0$ at the start of each round. Also by S_9, if v_i sends a *Coin* message to v_{i+1} then it increments its *coinCount*. Hence in each round, v_i can send at most n number of *Coin* messages. □

Lemma 2. *If* master *node* v_n *gets the (*end-of-round, $efac$*) message in a round and E is the number of triggers contributed by all nodes except* v_n *during the end-of-round phase then* $E \geq 0$.

Proof. By S_4, the master node v_n sends a (*end-of-round*, $efactor$) message to v_1 with $efactor = 0$. Any node v_i, on receiving a (*end-of-round*, $efactor$) message, computes its own contribution as $factor = C/\tau$. It then adds this $factor$ with the $efactor$ and forwards the (*end-of-round*, $efactor$) message to v_{i+1} with the modified $efactor$ value. Therefore eventually v_n will receive the (*end-of-round*, $efac$) message from v_{n-1}. Note here that the contribution of triggers

by v_n itself in this phase is done through the variable $C(v_n)$. Hence $E = efac \times \tau$. Since $efac \geq 0$ and $\tau \neq 0$ therefore $E \geq 0$. □

Lemma 3. *In each round, at least $\psi = w'/2$ number of triggers are counted by the system where w' is the remaining number of triggers yet to be counted at the begining of the round.*

Proof. By S_7 (or S_2) and S_8 (or S_3)) it is clear that $C(v_i)$ is updated when v_i receives a *Trigger* or a *Coin* message. Also by S_4, the master node v_n sends a (*end-of-round*, $efactor$) message to v_1 only if $fsorFlag(v_n) = true$ and $C(v_n) \geq \psi$. Again by Lemma 2, the number of triggers contributed by all nodes except v_n during the propagation of *end-of-round* message (from the time it is sent by v_n to the time it is received by v_n) is $E \geq 0$. Hence total number of triggers counted in a round is given by $C(v_n) + E$. Since $C(v_n) \geq \psi$, therefore the lemma is proved.

□

Theorem 1. *The distributed trigger counting takes $O(\log \frac{w}{n})$ rounds.*

Proof. Here we will analyze how the value of w is decreasing from one round to the next round. By Lemma 2, it depends on $E \geq 0$. Initially $w' = w$. By S_5, the remaining number of triggers to be counted in future rounds is given by $w' = w-(C(v_n)+E)$. Upper bound of the number of rounds can be obtained when $E = 0$. Here w' decreases as w, $\frac{1}{2}w$, $(\frac{1}{2})^2w$, $(\frac{1}{2})^3w$, ... , $(\frac{1}{2})^kw$. The number of rounds continue till $w' \geq 0$.

Let after $(r + 1)$ rounds $\tau < 1$ for the first time. So, in the r-th round $\tau \geq 1$. Therefore $\frac{(1/2)^r w}{n} \geq 1$. Therefore $r = O(\log \frac{w}{n})$. At $(r + 1)$-th round, $\tau \leq 1$. Hence $w'/2n \leq 1$. Hence $w' \leq 2n$. Since $\tau \leq 1$, if a node gets 1 trigger then its local threshold is crossed and a *Coin* message is sent. Therefore to count $w' \leq 2n$ number of triggers, at most $2n$ *Coin* messages will be required. In worst case this will require $2n$ rounds. Hence the total number of rounds is $O(2n + \log \frac{w}{n}) = O(\log \frac{w}{n})$ since $w >> n$.

□

Theorem 2 (Partial Correctness). *If the DTC algorithm has terminated then at least w triggers have been counted.*

Proof. Let us assume that the DTC algorithm has terminated. However at least w triggers have not yet been counted. Therefore by S_5, $w' > 0$. So v_n sets $sflag(v_n) = true$. Therefore G_1 becomes enabled. So eventually G_6 will be enabled. However this contradicts the assumption that the algorithm has terminated. Therefore $w' \leq 0$. Hence at least w triggers will be counted.

□

Theorem 3 (Termination). *The DTC algorithm for a ring network eventually terminates.*

Proof. By Lemma 1, each slave node sends at most n number of *Coin* messages in a round. After this, no slave will send a *Coin* message in the current round even if it crosses its local threshold or receives *Coin* message from its predecessor. Also if $C(v_n) \geq \psi$ then v_n sends a *end-of-round* message to v_1. Each slave node v_i, on receiving a *end-of-round* message, forward it to its successor. Eventually when v_n gets back a *end-of-round* message, it does not send any further *end-of-round* message in the current round. Here v_n checks is $w' > 0$. By Lemma 3, eventually $w' \leq 0$ will hold. Therefore v_n will set $sflag(v_n) = false$ and $fsorFlag(v_n) = false$. Hence G_1, G_2, G_3, G_4 are all $false$. Also by S_{11}, $fsorFlag(v_i) = false$ for any slave node v_i. Hence G_7, G_8, G_9 are $false$. Since G_1 is $false$, therefore v_n does not send *start-of-round* message. Here G_6 is $false$. Similarly G_{10} and G_{11} are also $false$. Therefore all the guards are $false$. Therefore the algorithm eventually terminates.

□

Theorem 4. *The message complexity of the DTC algorithm is $O(n^2 \log \frac{w}{n})$.*

Proof. In a round, each slave node can send n number of *Coin* messages. Also each of them is forwarded by the other slaves towards the master node. Hence in each round $O(n^2)$ number of messages are sent overall by all the nodes. By Theorem 1, the algorithm takes $O(\log \frac{w}{n})$ rounds. Hence the overall message complexity is $O(n^2 \log \frac{w}{n})$.

□

Theorem 5. *The MaxRcvLoad of the DTC algorithm in a ring is $O(n \log(w/n))$.*

Proof. In a round each node v_i receives $O(n)$ number of messages. Since the algorithm has overall $O(\log \frac{w}{n})$ rounds, the MaxRcvLoad per node is $O(n \log(w/n))$.

□

5 Tuning the Algorithm

The algorithm can be tuned by appropriately setting the parameters ψ and τ. In this case we have chosen $\tau = \psi/n$. If we choose $\tau = \psi$ then we may achieve a better message complexity and MaxRcvLoad. Let us assume that $\psi = w/k$ and $\tau = \psi/l$. Here we can claim that,

$$\begin{aligned} \textit{No of rounds} = \mathcal{R} &= O\left(\log_{\frac{k}{k-1}}(w/n)\right) \\ &= O\left(\log_{\frac{w}{w-\psi}}(w/n)\right) \end{aligned}$$

It can be observed that the round complexity is primarily dependent on ψ. This is due to the fact that one round completes only when all the nodes together counts a global threshold, ψ, number of triggers. The local threshold, τ, does not affect the number of rounds. The variation of the number of rounds with respect to ψ is shown in Figure 3. We consider three different values of w. It is observed that as ψ increases, the number of rounds required decreases.

The overall message complexity of the algorithm can be given as,

$$\mathcal{M} = O\left(\frac{n\psi}{\tau}\left(\log_{\frac{w}{w-\psi}}(w/n)\right)\right)$$

It is obvious from the above equation that $\mathcal{M}$ is directly proportional to $l = \psi/\tau$. In the proposed algorithm in this paper, $l = n$. If we assume $l = 1$ (i.e. $\psi = \tau$) then we can achieve a better overall message complexity of $O(n \log_{\frac{w}{w-\psi}}(w/n))$. Figure 4 depicts the variation of message complexity with respect to $l = \psi/\tau$. It is clear that as l increases the messages complexity increases. This is due to the fact that l can increse only when ψ is large and τ is small. In this case, each node crosses its local threshold (due

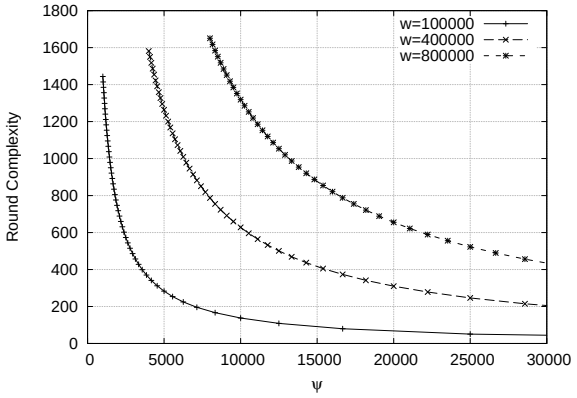


Figure 3: The variation of number of rounds with respect to ψ

to the receive of triggers) many more times and thus more number of *Coin* messages are sent in a round. This increases the overall message complexity of the system.

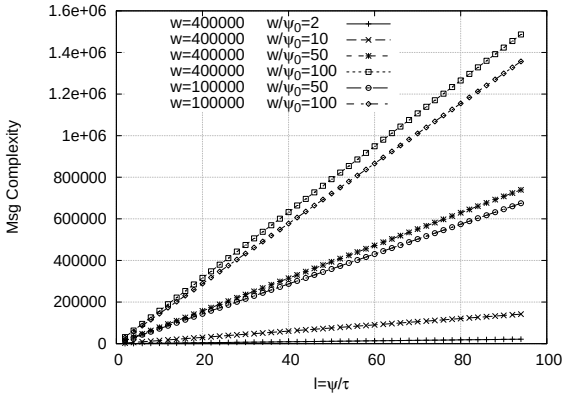


Figure 4: The variation of message complexity with respect to l

Similarly in Figure 5, we see the variation of message complexity with respect to τ. Here we keep w and ψ fixed and observe the variation of message complexity with respect to τ. It is found that as τ increases, message complexity falls sharply initially. However as the value of τ approaches ψ, the rate of fall in message complexity is small and not as sharp as earlier. This is an indication that we may not improve the message complexity by arbitrarily increasing the τ.

6 Message Complexity and Delay Trade-off

There is a trade-off between the message complexity of distributed trigger counting and the time of raising an alarm when at least w triggers have been counted. The simplest way to minimize the number of messages is to let every node detect triggers and when any one has detected w triggers, the global alarm signal is sent. However, in this simple scheme there is obviously a big risk that the system will be seriously delayed in sending the global alarm signal which has to be sent as soon as w or more triggers are received by all the nodes. The obvious way to avoid such delays is to send a message as soon as a trigger is detected.

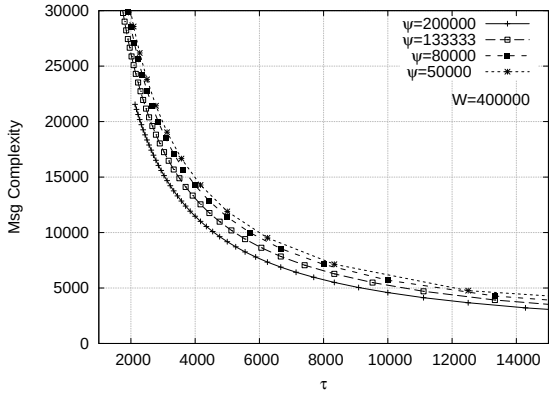


Figure 5: The variation of message complexity with respect to τ

However this will generate a lot of messages (potentially $O(w)$). Hence there should be some compromise between these two extremes.

Let t_0 be the time when w-th trigger enters the system (any node). Let t_1 be the time at which v_n sends the *TargetReached* message to v_1. We define the delay of generating the alarm as $\mathcal{T} = |t_1 - t_0|$. In the following we give a measure of $\mathcal{T}$.

Let e_1 be the $(w-1)$-th trigger and e_2 be the w-th trigger. Now there can be two cases:

Case 1: e_1 and e_2 enter the system in same round. Every trigger that enters the system is counted by the *master* node through *Coin* message or excess triggers during *end-of-round*. The maximum delay between the times e_1 and e_2 are recorded at the *master* node is the delay between the receive of two successive *Coin* messages by the *master* v_n. Essentially this is equal to the time required by τ triggers to arrive at any arbitrary node (where trigger count is zero). Let this time be denoted by Δ. In this case $\mathcal{T} = \Delta$. Here we assume that processing delay and communication delay is negligible.

Case 2: e_1 and e_2 enter the system in two different successive rounds. In this case, $\mathcal{T} = \Delta + \Delta_e$ where Δ_e is the amount of time required to collect all the excess triggers during the *end-of-round* phase.

If τ is large then delay will be large. However message complexity will be smaller. The reverse will happen if τ is small.

7 DTC in Arbitrary Network

To solve the DTC problem over an arbitrary network, we need to embed a logical unidirectional ring over the graph. For a certain class of graphs (Eulerian graphs), this can be done by forming an Eulerian circuit over the graph. Even though a Hamiltonian circuit provides the perfect ring embedding, the graph may not have an Hamiltonian circuit. Also finding a Hamiltonian circuit is NP-complete. Therefore our approach relies on Eulerian circuit construction. Makki (1999) proposed a distributed algorithm for construction of an Eulerian circuit. The algorithm has a message complexity of $(1 + r)(|\mathcal{E}| + n)$ where $0 \leq r < 1$. Here $\mathcal{E}$ denotes the set of edges of the graph. Our approach is to pre-process the input graph, provided it is Eulerian, to find an Eulerian circuit using the

above algorithm. Next on the virtual ring thus obtained, we apply the proposed trigger counting algorithm. Hence the overall message complexity of the DTC problem in an arbitrary network, which is Eulerian, is $O(|\mathcal{E}| + n^2 \log \frac{w}{n}) = O(n^2 \log \frac{w}{n})$. Also MaxRcvLoad for DTC in this case remains $O(n \log \frac{w}{n})$ since average message complexity of Eulerian circuit construction is $O(n)$.

8 Conclusion

We have presented a distributed algorithm for the DTC problem in a ring network with message complexity of $O(n^2 \log(w/n))$ and MaxRcvLoad of $O(n \log(w/n))$. One of the important issues is that the DTC algorithm may generate an alarm after counting w' number of triggers where $w' > w$. One of the future works may be to minimize the difference between w' and w. Proposing an algorithm for the DTC problem in an arbitrary network with optimal complexity can be a challenging work.

References

Awerbuch, B. (1985), 'Complexity of network synchronization', *Journal of ACM* **32**(4), 804–823.

Chakaravarthy, V. T., Choudhury, A. R., Garg, V. K. & Sabharwal, Y. (2011), An efficient decentralized algorithm for the distributed trigger counting problem, *in* 'Proceedings of the 12th International Conference on Distributed Computing and Networking', Springer-Verlag, Berlin, Bangalore, India, pp. 53–64.

Chakaravarthy, V. T., Choudhury, A. R. & Sabharwal, Y. (2011), Improved algorithms for the distributed trigger counting problem, *in* 'Proceedings of the 25th IEEE International Parallel and Distributed Processing Symposium', IEEE Computer Society, Washington, DC, USA, Anchorage, Alaska, USA, pp. 515–523.

Cormode, G., Muthukrishnan, S. & Yi, K. (2011), 'Algorithms for distributed functional monitoring', *ACM Trans. Algorithms* **7**(2), 1–20.

Emek, Y. & Korman, A. (2010), Efficient threshold detection in a distributed environment, *in* 'Proceedings of the 29th ACM Symposium on Principles of Distributed Computing (PODC)', ACM, New York, USA, Zurich, Switzerland, pp. 183–191.

Emek, Y. & Korman, A. (2011), 'New bounds for the controller problem', *Distributed Computing* **24**(3-4), 177–186.

Garg, R., Garg, V. K. & Sabharwal, Y. (2006), Scalable algorithms for global snapshots in distributed systems, *in* 'Proceedings of the 20th Annual International Conference on Supercomputing (ICS)', ACM, New York, USA, Cairns, Queensland, Australia, pp. 269–277.

Huang, L., Garofalakis, M., Joseph, A. D. & Taft, N. (2007), Communication-efficient tracking of distributed cumulative triggers, *in* 'Proceedings of the 27th International Conference on Distributed Computing Systems', IEEE Computer Society, Washington, DC, USA, Toronto, Canada, pp. 54–63.

Keralapura, R., Cormode, G. & Ramamirtham, J. (2006), Communication-efficient distributed monitoring of thresholded counts, *in* 'Proceedings of the ACM SIGMOD International Conference on Management of Data', ACM, New York, USA, Chicago, IL, USA, pp. 289–300.

Korman, A. & Kutten, S. (2007), Controller and estimator for dynamic networks, *in* 'Proceedings of the 26th ACM Symposium on Principles of Distributed Computing (PODC)', ACM, New York, USA, Portland, Oregon, USA, pp. 175–184.

Makki, S. A. M. (1999), 'Eulerian tour construction in a distributed environment', *Computer Communications* **22**(7), 621 – 628.

Defending Concealedness in IEEE 802.11n

Sandip Chakraborty, Subhrendu Chattopadhyay, Suchetana Chakraborty, Sukumar Nandi
Department of Computer Science and Engineering, IIT Guwahati, Assam, India 781039
Email: {c.sandip, subhrendu, suchetana, sukumar}@iitg.ernet.in

***Abstract*—IEEE 802.11 distributed coordination function supports two access mechanisms - the basic access and the four-way access, both of which are vulnerable to the hidden and the exposed node problem (collectively called the concealed node problem). Though most of the works in literature suggest to overlook this problem due to the associated overhead to solve them, this paper shows that the problem becomes severe for high speed wireless mesh networks. An opportunistic four-way access mechanism is designed to defend this problem in IEEE 802.11n mesh networks that supports high data rates. The performance of the proposed scheme is evaluated in a practical 802.11n indoor mesh testbed, that shows a significant performance improvement compared to the standard access mechanisms.**

***Keywords*-IEEE 802.11n; exposed; opportunistic access**

I. Introduction

IEEE 802.11 *'Carrier Sense Multiple Access with Collision Avoidance'* (CSMA/CA) supports two different access technologies - the *basic access*, where every data frame is followed by a corresponding acknowledgement (ACK) frame, and the *four-way access*, where two control frames are used to reserve channel before the actual data communication, namely the 'Request to Send' (RTS) and the 'Clear to Send' (CTS) frames. The basic access mechanism results in the well known *hidden node* problem [1]. Considering Fig. 1, both the nodes A and C want to transmit to the node B simultaneously. The basic access mechanism uses the concept of physical carrier sensing (PCS) before the data communication, where the nodes sense the channel for being idle. As node C is outside the carrier sense (CS) range of node A, it can not sense the ongoing data transmissions $A \rightarrow B$. As a result, node C may initiate another communication with node B, resulting interference near the receiver. The four-way access mechanism [2] solves the hidden node problem by communicating with RTS and CTS handshaking control frames before the actual data transmission. On overhearing these control frames, every node in the CS region of the transmitter and the receiver defer its transmissions to avoid interference. This procedure is called virtual carrier sensing (VCS). However, the four way access mechanism introduces another problem, called the *exposed node problem* [3]. Considering Fig. 1, a node G, outside the CS range of the receiver node B, can initiate communication from node F. However, on overhearing the CTS frame from node B, node F defers all the communication through it. Therefore, on receiving an RTS frame from node G, node F does not reply back with a CTS frame to initiate the communication. Here, F is a receiver-side exposed node, termed as *R-exposed* node in this paper, that can act as a potential receiver, but is blocked due to the VCS. Similarly, from Fig. 1, exposed node problem can also occur at the transmitter side because of the communication blockage due to RTS overhearing. These nodes are termed as *T-exposed* in this paper. In the figure, D is a T-exposed node that can also act as a potential transmitter for another receiver E, outside the CS range of the node B.

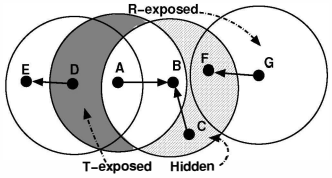


Fig. 1. Hidden, T-exposed and R-exposed nodes

The effects of the concealed nodes for IEEE 802.11 basic and four-way access mechanisms have been well studied in literature, such as [4] and the references therein. Recent researches have shown that RTS/CTS access mechanism does not perform well always, because of the signaling overhead [5] due to control message transmission. Nevertheless, the basic access mechanism suffers from both the hidden and the T-exposed nodes. On the contrary, the four-way access can solve the hidden node problem, but undergoes for both the T-exposed and R-exposed nodes. Though the exposed nodes reduce the spatial reuse, allowing communications to these nodes may result in data-ACK interference, if not properly synchronized. However, as shown in this paper, the exposed node problem becomes severe in the case of high data rate mesh networks built upon the 802.11n technology. In [6], the authors have shown that RTS/CTS exchange can results in performance drop for high data rate mesh networks by increasing number of exposed nodes. However, they have not considered the complete features of 802.11n high speed networks, such as frame aggregation and block ACK (BACK). A number of works exist for mitigating the exposed terminal problem, such as [3] and the references therein, however, they require even additional messages over RTS/CTS, and do not consider the advanced technologies supported by 802.11n. This paper theoretically models the performance of the high data rate mesh networks for three different scenarios with concealed nodes to show their effect over the high speed network

The works of Sandip Chakraborty and Suchetana Chakraborty are supported by the TATA Consultancy Services (TCS), India through the TCS Research Fellowship Program. The authors would like to thank Mr. Rajendra Singh, Skiva Technologies for providing necessary hardware and technical supports to implement the testbed.

978-1-4799-3635-9/14/$31.00 ©2014 IEEE

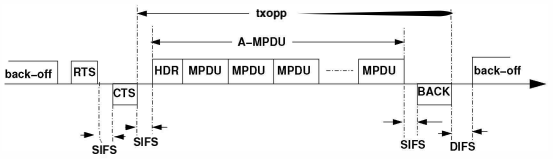


Fig. 2. 802.11n frame aggregation

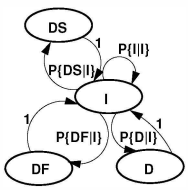


Fig. 3. Basic Access

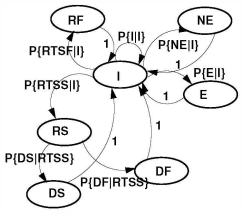


Fig. 4. Four-way Access

performance. An opportunistic RTS/CTS access mechanism is designed to mitigate the problem with the hidden nodes and T-exposed nodes completely, and to partially solve the problem with R-exposed nodes. The frame aggregation and BACK techniques for 802.11n network are used for avoiding data-ACK interference at the exposed nodes. The performance of the proposed scheme is analyzed through the experimental results from a practical 802.11n indoor mesh testbed.

II. Modeling Concealedness in 802.11n

802.11n [7] can support data rates upto 600 Mbps by employing full-duplex dual band multiple input multiple output (MIMO) technologies. It uses the optional four-way handshake mechanism along with the *'frame aggregation'* and BACKs. In the frame aggregation scheme, several MAC protocol data units (MPDUs) are aggregated in a common data unit, called the A-MPDU, associated with a common header. Once the A-MPDU is received, a BACK is forwarded, that contains the MPDU sequence numbers, which are not received correctly due to interference, and need to be retransmitted. If the BACK is not received, the entire A-MPDU is assumed to be lost. The four-way access mechanism for 802.11n frame aggregation and BCAK is shown in Fig. 2. SIFS and DIFS denote the short inter-frame space and data inter-frame space durations, respectively.

A. System Model

This paper considers a network with n nodes uniformly deployed in a plane based on a homogeneous spatial Poisson distribution with node density λ. Every node follows contention based channel access through IEEE 802.11 distributed coordination function (DCF) employing a binary exponential back-off procedure. Let R_{CS} denote the CS range, and R_{IF} denote the interference range. In general, $R_{CS} \leq R_{IF}$ Every node supports 802.11n frame aggregation with BACK capabilities. Let β denote the aggregation level of every node, that means, an A-MPDU can contain β number of MPDUs. For the purpose of theoretical modeling, a discrete time model is considered where the time is divided into fixed length slots of length σ μs, and every event occurs at the beginning of a time slot. The data frames are generated at every node based on a Pareto-normal distribution with a minimum data generation duration as ω μs. Let α denote the probability that a node has data to transmit at time duration $\mathcal{T}$. Then, $\alpha = (\omega/\mathcal{T})^c$, where c is a constant. It can be noted that, if otherwise mentioned, the terms 'transmitter' and 'receiver' denote a data transmitter and a data receiver, respectively.

B. Basic Access with Hidden and T-exposed Nodes

The 802.11n basic access is represented as a 4-state discrete time Markov model, as shown in Fig 3. In a time slot, a node in the system can be in one of the four states;

(i) *Idle State (I):* A node remains in the idle state, if it does not have data to transmit. The average time duration of this state (T_I) is, $T_I = (1-\alpha)\sigma$.

(ii) *Data Success State (DS):* This state represents the successful transmission of data. Let H, D_M and T_a denote the A-MPDU header size, the MPDU size, and the expected A-MPDU transmission time, respectively. Therefore $T_a = H + \alpha\beta D_M$. Assume ρ be the propagation delay, $SIFS$ and $DIFS$ be the two inter-frame space times, and T_b be the BACK transmission time. Therefore, the duration of this state (T_{DS}) is calculated as;

$$T_{DS} = T_a + SIFS + \rho + T_b + SIFS + \rho$$

(iii) *Data Failure State (DF):* This state represents the failure of data transmission due to the interference. As the data and the BACK communications are in reverse directions, the interference is of two types - the data-data interference, and the data-BACK interference. For the data-data interference, the average number of corrupted MPDUs is equal to $\beta\alpha^2$. The data-BACK interference can corrupt either one or two MPDUs, making an average loss of 1.5 MPDUs. Further, if the BACK is lost due to the data-BACK interference, the complete A-MPDU is considered to be lost. Let $T_{ab} = T_a + T_b$. Assuming T_{DF} is the average time a node remains in the DS state,

$$T_{DF} = \frac{T_a}{T_{ab}}(0.5\alpha^2\beta D_M + 0.5 \times 1.5 D_M) + \frac{T_b}{T_{ab}} T_{DS}$$

(iv) *Deferred State (D):* A node goes to the deferred state due to the PCS. The duration of this state is $T_D \approx T_{DS}$.

1) Calculation of Steady State Probabilities: Let $\mathcal{P}_I$, $\mathcal{P}_{DS}$, $\mathcal{P}_{DF}$ and $\mathcal{P}_D$ denote the steady state probabilities for the states I, DS, DF and D, respectively. Then considering the time-homogeneity, the steady state probabilities for the Markov model, shown in Fig. 3, are expressed as follows.

$$\mathcal{P}_{\mathbb{S}_b} = \mathcal{P}_I \times \mathrm{P}\{\mathbb{S}_b | I\}; \qquad \mathbb{S}_b \in \{DS, DF, D\} \tag{1a}$$

$$\mathcal{P}_I = \mathcal{P}_I \times \mathrm{P}\{I|I\} + \mathcal{P}_{DS} + \mathcal{P}_{DF} + \mathcal{P}_D \tag{1b}$$

Let τ denote the probability that a node attempts for a data transmission. The steady state value of τ is calculated according to [8] as; $\tau = 2/(CW+1)$ where CW is the

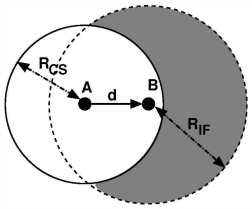


Fig. 5. Basic access

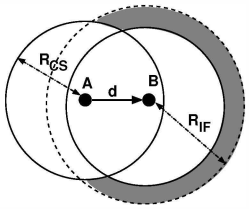


Fig. 6. Four-way access

average contention window size used in IEEE 802.11 DCF back-off procedure. Therefore;

$$\tau = \mathrm{P}\{DS|I\} + \mathrm{P}\{DF|I\} \tag{2}$$

Substituting the values and simplifying, $\mathcal{P}_I$ is represented as;

$$\mathcal{P}_I = 1/(1 + \tau + \mathrm{P}\{D|I\}) \tag{3}$$

The average duration of a transmission opportunity, denoted as $\mathcal{T}$, is calculated as;

$$\mathcal{T} = \mathcal{P}_I T_I + \mathcal{P}_{DS} T_{DS} + \mathcal{P}_{DF} T_{DF} + \mathcal{P}_D T_D \tag{4}$$

2) Analysis of Data Transmission: Let $\mathcal{E}_1$ denote the event that no node in an area Ψ transmits in a given time slot. Based on the Poisson distribution of the nodes, the probability of the event $\mathcal{E}_1$ is calculated as,

$$\wp(\mathcal{E}_1) = \sum_{\nu=0}^{\infty} (1-\tau)^{\nu} \frac{(\lambda\Psi)^{\nu}}{\nu!} e^{-\lambda\Psi} = e^{-\tau\lambda\Psi} \tag{5}$$

Considering Fig. 5, the shaded region indicates the area where hidden nodes can exist for the transmission$A \rightarrow B$. The area of the shaded region, denoted as $\Psi_h(d)$, is given by,

$$\Psi_h(d) = \pi R_{IF}^2 - \Psi_l(R_{CS}, R_{IF}, d) \tag{6}$$

where the value of $\Psi_l(R_{CS}, R_{IF}, d)$ is calculated using simple geometrical analysis. Assuming a spatial Poisson distribution of the nodes, let $f_{CS}(d)$ denote the probability density function (PDF) of the distance between nodes A and B. $f_{CS}(d)$ is calculated as;

$$f_{CS}(d) = \frac{2\pi d}{\pi R_{CS}^2} = \frac{2d}{R_{CS}^2} \tag{7}$$

$\mathrm{P}\{DS|I\}$ and $\mathrm{P}\{DF|I\}$ are calculated as;

$$\mathrm{P}\{DS|I\} = \int_0^{R_{CS}} e^{-\tau\lambda\Psi_h(x)} f_{CS}(x) dx \tag{8}$$

$$\mathrm{P}\{DF|I\} = 1 - \mathrm{P}\{DS|I\} \tag{9}$$

3) Analysis of Deferred State: A node goes to the deferred state if any other node within its CS region starts transmission. In case of the basic access, the nodes in the deferred state include the T-exposed nodes. Henceforth, $\mathrm{P}\{D|I\}$ is calculated as;

$$\mathrm{P}\{D|I\} = \int_0^{R_{CS}} \left(1 - e^{-\tau\lambda\pi R_{CS}^2}\right) f_{CS}(x) dx \tag{10}$$

C. Four-way Access with Exposed Nodes

Fig. 4 presents a 7-state Markov model to analyze the four-way access mechanism. This mode introduces two new states - *RTS success (RS)* and *RTS failure (RF)*, and divides the deferred state into two states - *exposed (E)*, and *non-exposed (NE)*, that is the deferred nodes that are not exposed. The data success (DS) and the data failure (DF) states are initiated after the RS state. Let T_{RTS} and T_{CTS} denote the RTS and the CTS transmission times, respectively, and $T_{\mathcal{S}}$ denote the duration of state $\mathcal{S}$. Then, the duration of these states are calculated as follows,

$$T_{RS} = T_{RTS} + SIFS + \rho + T_{CTS} + DIFS + \rho$$
$$T_{RF} \approx T_{RS};\ T_{NE} \approx T_{DS} + T_{RTS};\ T_E \approx T_{NE}$$

The duration of the DS and DF states are similar to the basic access scenario.

1) Calculation of Steady State Probabilities: Let $\mathcal{P}_I$, $\mathcal{P}_{RS}$, $\mathcal{P}_{RF}$, $\mathcal{P}_{DS}$, $\mathcal{P}_{DF}$, $\mathcal{P}_{NE}$ and $\mathcal{P}_E$ denote the steady state probabilities for the states I, RS, RF, DS, DF, NE and E, respectively. Then considering the time-homogeneity, the steady state probabilities for the Markov model, shown in Fig. 4, are expressed as follows.

$$\mathcal{P}_{\mathbb{S}_f} = \mathcal{P}_I \times \mathrm{P}\{\mathbb{S}_f|I\};\ \mathbb{S}_f \in \{RF, RS, NE, E\} \tag{11a}$$

$$\mathcal{P}_{\mathbb{S}_{f'}} = \mathcal{P}_{RS} \times \mathrm{P}\{\mathbb{S}_{f'}|RS\};\ \mathbb{S}_{f'} \in \{DS, DF\} \tag{11b}$$

$$\mathcal{P}_I = \mathcal{P}_I \times \mathrm{P}\{I|I\} + \mathcal{P}_{RF} + \mathcal{P}_{DS} + \mathcal{P}_{DF} + \mathcal{P}_{NE} + \mathcal{P}_E \tag{11c}$$

Similar to the basic access mechanism, the average duration of a transmission opportunity for the four-way access, denoted as $\mathcal{T}_f$, is represented as;

$$\mathcal{T}_f = \mathcal{P}_I T_I + \mathcal{P}_{RS} T_{RS} + \mathcal{P}_{RF} T_{RF} + \mathcal{P}_{DS} T_{DS} + \mathcal{P}_{DF} T_{DF} + \mathcal{P}_{NE} T_{NE} + \mathcal{P}_E T_E \tag{12}$$

2) Analysis of RTS/CTS Communication: The analysis of the RTS/CTS communication is similar to the analysis of the data transmission for basic access, except that the communication is successful if both the RTS and the CTS are transmitted successfully. Therefore, during RTS (or CTS) transmission, no other node in the interference range of the RTS (or CTS) transmitter should initiate another transmission. Let $\mathcal{E}_3$ denotes the event that no other node in the interference range of the transmitter and the receiver initiates a transmission. The probability of this event, denoted as $\wp(\mathcal{E}_3)$, is derived as;

$$\wp(\mathcal{E}_3) = e^{-2\tau\lambda\Psi_{RC}(R_{IF}, d)} \tag{13}$$

where $\Psi_{RC}(R_{IF}, d)$ denote the area bounded by the interference range of the RTS/CTS transmitter where the nodes are in d distance apart, and can be calculated using a similar procedure. As a result,

$$\mathrm{P}\{RS|I\} = \int_0^{R_{CS}} e^{-2\tau\lambda\Psi_{RC}(R_{IF}, x) f_{IF}(x) dx} \tag{14}$$

$$\mathrm{P}\{RF|I\} = 1 - \mathrm{P}\{RS|I\} \tag{15}$$

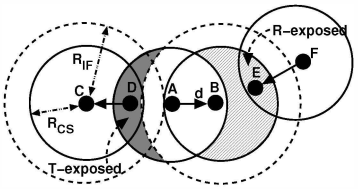


Fig. 7. Exposed nodes for four-way access

where $f_{IF}(x)$ is obtained as, $f_{IF}(x) = \frac{2x}{R_{IF}^2}$.

3) Analysis of Data Communication: In case of the four way handshaking, data failure can occur if a node within the interference range, but outside the CS range of the receiver, initiates a transmission. This can happen because of $R_{IF} > R_{CS}$. Assuming Fig. 6, data is transmitted successfully, if no node in the shaded region initiates a transmission. Let $\Psi_{data}(d)$ denotes the area of the shaded region, when the nodes are in d distance apart. The value of $\Psi_{data}(d)$ can be calculated using s similar procedure.

$$\mathrm{P}\{DS|RS\} = \int_0^{R_{CS}} e^{-2\tau\lambda\Psi_{data}(x)f_{CS}(x)dx} \tag{16}$$

$$\mathrm{P}\{DF|RS\} = 1 - \mathrm{P}\{DS|RS\} \tag{17}$$

4) Analysis of the Deferred Scenarios: During RTS/CTS communication, all the nodes in the CS ranges of both the transmitter and the receiver will defer their communications. As discussed earlier, these nodes are grouped into two classes - the non-exposed nodes, and the exposed nodes, including both the T-exposed and the R-exposed. The shaded regions in Fig. 7 show the position of the exposed nodes when the communicating nodes are in d distance apart. The area of this region, denoted by $\Psi_E(d)$, is calculated as;

$$\Psi_E(d) = 2\pi R_{CS}^2 - \Psi_l(R_{CS}, R_{IF}, d) - \Psi_{RC}(R_{CS}, d) \tag{18}$$

Similarly, The area of the region where non-exposed nodes are placed, denoted by $\Psi_{NE}(d)$, is calculated as;

$$\Psi_{NE}(d) = \pi R_{IF}^2 - \Psi_l(R_{CS}, R_{IF}, d) \tag{19}$$

Therefore,

$$\mathrm{P}\{E|I\} = \int_0^{R_{CS}} e^{-2\tau\lambda\Psi_E(x)f_{CS}(x)dx} \tag{20}$$

$$\mathrm{P}\{NE|I\} = \int_0^{R_{CS}} e^{-2\tau\lambda\Psi_{NE}(x)f_{CS}(x)dx} \tag{21}$$

D. Calculation of the Per-Node Throughput

The per-node throughput, represented as $\mathcal{G}$, is calculated as,

$$\mathcal{G} = \frac{\mathcal{P}_{DS}T_{DS}}{\mathcal{T}} \tag{22}$$

TABLE I
NUMERICAL VALUES OF THE MODELING PARAMETERS

Parameter	value	Parameter	value
MPDU size	1024 bytes	CTS size	14 bytes
β	20	Data rate	300 Mbps
BACK size	20 bytes	Slot time	$10\mu s$
A-MPDU header	28 bytes	SIFS	$10\mu s$
RTS size	20 bytes	DIFS	$50\mu s$

E. Model Verification and Analysis

This subsection verifies the proposed theoretical model, and analyzes the performance of the 802.11n mesh network for the different scenarios based on the numerical data obtained from the proposed theoretical model. The parameters used for the calculation of the theoretical model is summarized in table I. The data sets are generated using Maxima toolbox for the three different access models based on the theoretical analysis - the basic access, the four-access and the optimal access. The model for the optimal access scenario is derived from the four-way access model, shown in Fig. 4, by assuming $\mathcal{P}_E = 0$. This indicates that the exposed nodes are allowed to transmit with the assumption that interference does not occur.

1) Verification of the Theoretical Model: To verify the theoretical model proposed in this section, the numerical data obtained from the model is compared with the simulation results. The simulation is performed in NS-2.35 network simulator framework for both the basic access and the four-way access mechanism for 802.11n with frame aggregation and BACK support. Similar parameters, as given in table I, are used for the simulation purpose. The results are plotted with respect to the data generation probability (α) at every node. Every simulation setup is executed for 10 times with different seed values, and the average is taken to plot the graphs. The confidence interval, expressed in terms of the difference between the maximum and the minimum results, are also shown in the graphs. For both the theoretical and the simulation results, node density is considered as 16 nodes per $100m$, where the mean carrier sensing range is $50m$ with $5m$ variance. In the simulation setup, the capture threshold is taken as $20dB$ with transmit power $16dBm$ and receiver sensitivity $-85dBm$, according to RT-3352 802.11n wireless routers [9] (used for the testbed setup, as discussed in subsequent section). With this setup, the mean interference range becomes $65m$ with $5m$ variance.

Fig. 8 and Fig. 9 show the comparisons between the theoretical and the simulation results for the basic access and the four-way access, respectively. The vertical lines in the simulation results show the confidence interval. Both the figures show that the theoretical results and the simulation results are similar.

2) Effect of Hidden and Exposed Nodes: Fig. 10 compares the normalized throughput with respect to α for three scenarios, as discussed earlier. The figure reveals that when the data generation rate is very low, basic access performs marginally better than the four-way access. This is because of

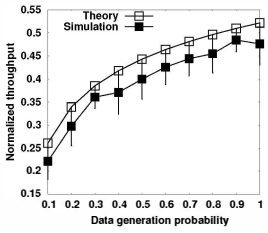


Fig. 8. Basic access

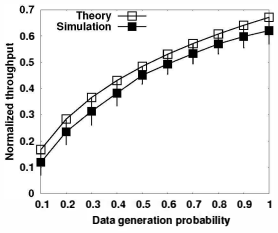


Fig. 9. Four-way access

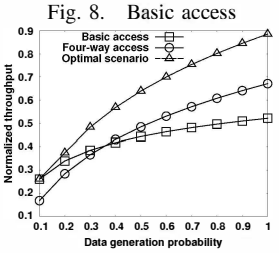


Fig. 10. Effect of α

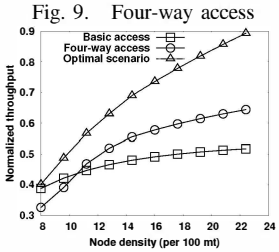


Fig. 11. Effect of λ

the signaling overhead associated with RTS/CTS handshaking. However, with higher data generation rates, the four-way handshake performs better than the basic access. The network throughput can be considerably improved with the optimal channel access, where the concealed nodes are completely avoided. Similar result is obtained from Fig. 11, that shows the normalized throughput with respect to the node density λ. As λ value increases, more exposed and hidden nodes are generated in the system, resulting in performance degradation for the basic access and the four-way access. Further, the signaling overhead associated with the optimal scenario is considerably lower than the benefits even in case of the low load scenario. Both the figures reveal that the optimal scenario provides better performance compared to the basic access and the four-way access, for both the low-load and the high-load scenario. Therefore, it is desirable to eliminate the concealed nodes as much as possible to improve the network performance by extensive spatial reuse.

III. Opportunistic Access: Design and Analysis

In the proposed opportunistic access mechanism built over four way access, the RTS and CTS messages are used for the detection of hidden nodes, as well as transmission is allowed to the exposed nodes. The data-BACK interference at T-exposed nodes are solved using a mechanism called 'null framing'. However, the interference at R-exposed nodes are not avoidable, and therefore the proposed opportunistic access mechanism allows transmission for the R-exposed nodes only if the data loss due to interference is comparatively less than the performance gain. The detailed design of the opportunistic access mechanism is discussed in following sections.

A. Opportunistic RTS/CTS Access

The legacy RTS/CTS protocol uses a table, called the *network allocation vector* (NAV), to maintain the transmission blockage on overhearing the RTS/CTS frames. The RTS/CTS frames contain a duration field (DU) that indicates the time duration for the channel reservation. On overhearing the RTS/CTS frames, every node sets its NAV for the time

Algorithm 1 Node S wants to transmit data to node R

1: **if** $(CTS_{act} = NULL) \wedge (\forall RTS_{act}.DST \notin \mathcal{N}_S)$ **then**
2: Send RTS; /*T-exposed nodes*/
3: **else**
4: Back-Off and retry;
5: **end if**

Algorithm 2 Node R receives RTS from node S

1: **if** $(RTS_{act} = NULL) \wedge (\forall CTS_{act}.DST \notin \mathcal{N}_R)$ **then**
2: Send CTS; /*R-exposed nodes*/
3: **end if**
4: $RTS_{act} \leftarrow this.RTS$ /*Append the received RTS in RTS_{act}*/

mentioned in the DU field. In the proposed augmentation of four-way access, different NAVs are maintained for every overheard RTS/CTS frame to detect the exposed nodes, and to avoid the data-BACK and data-control interferences. Let, RTS_{act} and CTS_{act} denote the sets of nodes from which a node has received the RTS or CTS frames, respectively, and $\mathcal{F}.DST$ denote the destination address associated with frame $\mathcal{F}$. Further assume that $\mathcal{N}_i$ denote the set of nodes that are in the CS range of node i. This set can be populated using standard beaconing.

The decision controls for exposed node detection are explained through Algorithm 1 and Algorithm 2, as described in the following subsections.

1) Detection of Hidden Nodes: The hidden node scenario can occur for two cases - (i) a transmitter node is within the CS range of another receiver node, and (ii) a receiver node is within the CS range of another transmitter node. For the first scenario, considering Fig. 1, let $A \rightarrow B$ communication starts first. Therefore, node C can overhear the CTS frame from node B, which is included in the CTS_{act} set. According to Algorithm 1, the condition is evaluated to be false, as $CTS_{act} \neq NULL$, and node C starts the back-off without initiating the communication.

For the second scenario, let us assume that node C is outside the CS range of node B, but within the CS range of node A. In this scenario node C acts as a receiver, and therefore, should not initiate a communication. Node C can overhear the RTS from node A, and includes it in the RTS_{act} set. On receiving the RTS frame from any other node, say H, node C executes Algorithm 2. However, the condition is evaluated to be false as $RTS_{act} \neq NULL$. Therefore it does not replies back with the CTS, and the communication is deferred.

2) Spatial Reuse by T-exposed Nodes: Considering Fig. 1, let $A \rightarrow B$ communication starts first. Node D within the CS range of node A wants to initiate a transmission with node E, which is outside the CS range of node A. As node D is outside the CS range of node B, it does not receive any CTS. Further, $RTS_A.DST \notin \mathcal{D}$. Therefore, the condition in Algorithm 1 is evaluated to be true, and node D forwards the RTS to node E. Node E is outside the CS range of both the nodes A and B. Therefore, the condition of Algorithm 2 is also evaluated to be true, and E replies back with the CTS, resulting in communication initialization.

3) Spatial Reuse by R-exposed Nodes: Considering Fig. 1, let $A \rightarrow B$ communication starts first. As node F is outside the CS range of node A, it does not overhear the RTS. Therefore $RTS_{act} = NULL$. Assume, node F receives an RTS from node G. As node F is within the CS range of node B, it overhears the CTS from node B. However, $CTS_B.DST = A \notin \mathcal{N}_F$. Therefore following Algorithm 2, it replies back with the CTS, resulting in the communication initialization.

B. Interference resolution

As discussed earlier, allowing the communication to the exposed nodes may result in data-BACK or data-control interferences. Two main properties of 802.11n access mechanism is explored to solve this problem, the frame aggregation along with BACK, and multiple simultaneous data streaming using MIMO technology. The MIMO with dual streaming in 802.11n allows a node to transmit and receive simultaneously using two different channels (20/40 MHz).

Interference can not be avoided completely for the nodes that are outside the CS range of the receiver. A node can avoid interference by overhearing the existing communications within its CS range. Further, the interference can occur either at the transmitter side (when there is an ongoing communication within the transmitter's CS range) or at receiver side (when there is an ongoing communication within the receiver's CS range). Based on this observation, either transmitter or receiver takes the responsibility to avoid interference, as described in the following subsections.

1) Sender Side Interference Avoidance: Considering Fig. 1, allowing $D \rightarrow E$ communication simultaneously with $A \rightarrow B$ communication may result in data-control and data-BACK interference at D, when E sends the CTS and the BACK frames, respectively. Similarly data-BACK interference can occur at A, when B sends the BACK frame. The control frames are lost in this scenario. To avoid this situation, this paper introduces the concept of NULL framing within an A-MPDU frame, as shown in Fig. 12. The NULL framing is a transmission gap within the A-MPDU frames, that indicates a possible interference duration. As shown in Fig. 12, on overhearing the RTS frame from node D, node A forwards a NULL frame after the current MPDU. This information is also appended in the MPDU sub-header to inform the receiver B about the upcoming NULL frame. To avoid interference, the size of the NULL frame is made equal to $SIFS + T_{CTS} + DIFS$. Further, node D initiates the RTS transmission at the end of a $DIFS$ interval, after it senses the beginning of an MPDU transmission from A. Similarly, from the DU field of the overheard RTS from node A, node D knows the end of the reservation period for node A. Therefore, node D forwards another NULL frame to avoid the data-BACK interference.

2) Receiver Side Interference Mitigation: Considering Fig. 1, let $A \rightarrow B$ starts earlier than $G \rightarrow F$, resulting in data-CTS and data-BACK interference at nodes B and F. The data frames will be lost in this scenario. However, as the size of the control and BACK frames are less than the MPDU, either one or two MPDUs will be lost. Further, the transmission of data frames are controlled by the transmitter, and in this scenario, the coordination between the transmitters is costly as they are more than two hops away. Therefore, interference can not be avoided completely. In the proposed opportunistic access, the receiver initiates the communication if the losses of MPDUs due to the interference is less. Based on the CTS overhearing, node F can determine the number of receivers within its CS range. Let the number of such receivers be $\Re_F$. Due to the data-CTS and data-BACK interference, maximum number of MPDU losses can be calculated as $4 \times \Re_F$. Let $G \rightarrow F$ communication wants to reserve the channel for $P_{G \rightarrow F}$ numbers of MPDUs. Then, node F initiates the communication by replying back with a CTS frame, if $(4 \times \Re_F / P_{G \rightarrow F}) < \Upsilon$, where Υ is a constant threshold. The value of Υ is determined based on the cost-benefit trade-off, as analyzed in the next section.

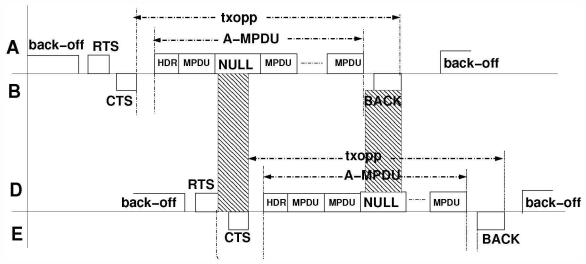


Fig. 12. Interference scenario and NULL framing

To find out the expected value of $\Re_i$ for a node i, let us assume that ϑ be the probability that a node acts as a receiver. From Fig. 1, the nodes that are inside the CS range of both the transmitter and the receiver, can not act as a receiver due to the hidden node problem. Therefore, only the nodes that are in the CS range of the receiver, but not in the CS range of the transmitter, can act as a receiver. Let Ψ_R denote the area where a receiver node can exist. Ψ_R can be calculated as,

$$\Psi_R = R_{CS}^2 (\frac{2\pi}{3} + \frac{\sqrt{3}}{2})$$

From the assumption of the spatial Poisson distribution of the nodes, the expected number of receiver within the CS region of a node i, denoted as $E[\Re_i]$, is calculated as,

$$E[\Re_i] = \vartheta \sum_{\nu=0}^{\infty} \nu \frac{(\lambda \Psi_R)^\nu}{\nu!} e^{-\lambda \Psi_R} = \vartheta \lambda \Psi_R \qquad (23)$$

From the steady state distribution, $\vartheta = \alpha$, where α is the frame generation probability as mentioned earlier. Henceforth;

$$E[\Re_i] = \alpha \lambda R_{CS}^2 (\frac{2\pi}{3} + \frac{\sqrt{3}}{2}) = 2.96 \alpha \lambda R_{CS}^2 \qquad (24)$$

Let there are ε number of nodes in the area πR_{CS}^2 (the area bounded by the CS range of a node). Therefore,

$$E_{MAX}[\Re_i] = 2.96 \alpha \frac{\varepsilon}{\pi R_{CS}^2} R_{CS}^2 = 0.94 \alpha \varepsilon \qquad (25)$$

As $\alpha \leq 1$, the expected number of interfering frames is considerably lower than the number of frames to be transmitted by a R-exposed node. Therefore, in spite of some frame losses,

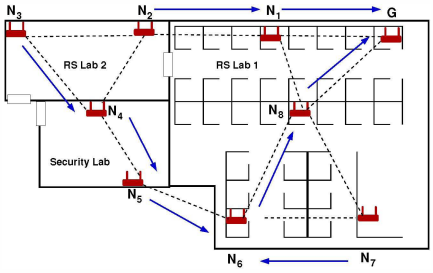


Fig. 13. 802.11 Indoor mesh testbed and connectivity layout

the overall performance of the network can be improved by allowing the transmission to the R-exposed nodes.

IV. Experimental Results

This section analyzes the performance of the proposed opportunistic access through the results from a indoor mesh testbed.

A. Testbed Analysis

The proposed scheme is implemented and evaluated using a 9-node 802.11n indoor mesh testbed deployed over the IIT Guwahati computer science department research labs, as shown in Fig. 13. The connectivity among the nodes is shown using dotted lines. The node G works as the mesh gateway. Each node is a Skiva Easyconnect RT001 N300 WiFi router with RaLink RT-3352 chipset [9]. This chip can support up to 300 Mbps data rate with maximum transmission power of 200 mW, that corresponds to average $10m$ CS range and $15m$ interference range, with a deviation of $\pm 5m$ based on the external noise factor.

1) Testbed Setup: Every node communicates with the gateway through multi-hop communication. On average 10 number of clients are attached with every mesh node. Trivial File Transfer Protocol (TFTP) is used as the application layer traffic, that uses UDP at the network layer. Two scenarios have been evaluated - a scenario when the network is loaded, and in the second scenario, the network load is low. In the fully loaded scenario, every client generates traffic at an average rate of 10 Mbps. The average traffic generation rate for the lightly loaded scenario is 1 Mbps. The proposed opportunistic access mechanism is implemented as a loadable kernel module (LKM) at every router. The MAC layer calls the LKM to decide the access policy. The blue arrows in Fig. 13 indicate an example data flow communication architecture in the mesh network that is used for evaluation purpose. It can be noted that the MAC layer forwarding decision is based on a link quality metric, called the expected transmission count [10], that periodically measures the link quality in terms of signal strength and frame loss. Therefore, the best path is not necessarily the minimum-hop path. The performance of the proposed opportunistic access scheme is compared with the basic access and the four-way access mechanisms. For 802.11n, dual streaming is used with frame aggregation level set to 20. The MPDU size is kept fixed at 256 Kb.

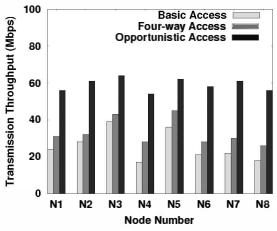


Fig. 14. Throughput: High Load

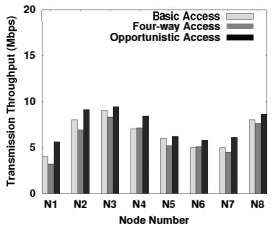


Fig. 15. Throughput: Low Load

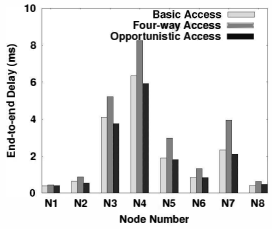


Fig. 16. Delay: High Load

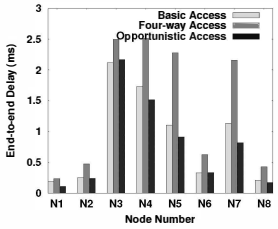


Fig. 17. Delay: Low Load

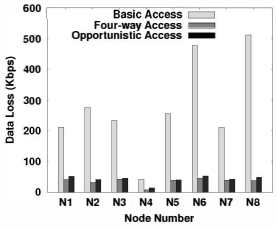


Fig. 18. Loss: High Load

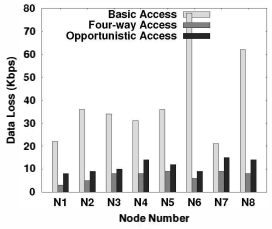


Fig. 19. Loss: Low Load

2) Per Node Performance: Fig. 14 and Fig. 15 compare the performance of the opportunistic scheme with other naive approaches in terms of average transmission throughput per mesh node. The average transmission throughput is calculated as the amount of data transmitted successfully per unit time. For the fully loaded network, the transmission throughput for the four-way access is more than the transmission throughput for the basic access, as the four way access reduces data loss due to hidden nodes. On the contrary, Fig. 15 reveals that for the lightly loaded network, basic access performs better than four-way access, which is because of extra control overhead. However, both the figures show that the proposed opportunistic access mechanism performs better than other two approaches, for both the fully loaded network and the lightly loaded network.

Fig. 16 and Fig. 17 show the forwarding delay for the above two scenarios. The figure reveals that for both the fully loaded and the lightly loaded scenario, the forwarding delay for the four-way access is more than the basic access. The analysis of the individual packet traces have revealed that this increment forwarding delay is partially because of RTS/CTS handshaking, and more because of the transmission deferring due to the VCS. It can be noted that due to the frame aggregation and BACK schemes, the RTS/CTS handshaking overhead is considerably reduced in case of 802.11n. Further, the transmission deferring is avoided in the case of the proposed opportunistic forwarding by allowing exposed node to participate in communications, which decreases the forwarding delay substantially by reducing the waiting time.

Fig. 18 and Fig. 19 analyze the packet loss for the three

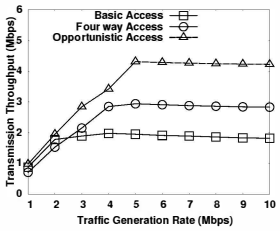


Fig. 20. Throughput

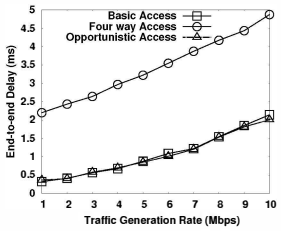


Fig. 21. Forwarding Delay

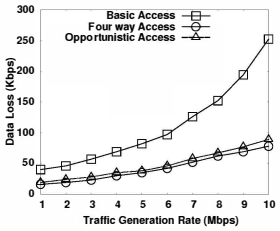


Fig. 22. Data Loss

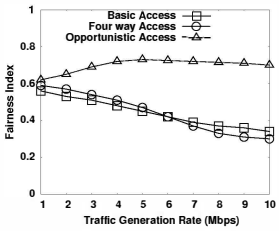


Fig. 23. Fairness

access protocols in the two different scenarios. These figures divulge that the data loss is considerably more in the case of basic access mechanism. The reason behind the data loss for the basic access is mainly due to the hidden nodes. For the testbed scenario, number of hidden nodes is more for nodes $N6$ and $N8$, which results in high data loss for these two nodes, as seen from Fig. 18. For the fully loaded network, the amount of data loss is significant, compared to the four-way access and the opportunistic access. The four way access and the opportunistic access avoids the data loss by reserving the channel through the RTS/CTS handshaking and the VCS. The small data loss for the four way access and the opportunistic access is due to the interference from the nodes which are outside the CS range.

3) Effect of the Network Traffic Load: For this set of experiments, the performance of the individual clients are evaluated by varying traffic generation rates. Fig. 20 shows the average per client transmission throughput with respect to the traffic generation rate. As usual, the performance of the opportunistic access is more than the basic access and the four-way access. One interesting observation can be made from the figure that the saturation point of the network is different for different channel access mechanism. While the basic access makes the network saturated at per client 2 Mbps data generation rate, the four-way access saturates the network as 4 Mbps, and the opportunistic access makes the network saturated at 5 Mbps. Because of the hidden nodes, lots of data frames are dropped, which reduces the network capacity for the basic access. The T-exposed nodes further reduces the capacity by blocking the nodes through the PCS. Though the hidden node problem is being solved for the four-way access, the exposed node problem causes the reduction in network capacity for the four-way access mechanism. The proposed opportunistic access mechanism unleash the network capacity by solving both the concealed node problem. Fig. 21 and Fig. 22 show the average end-to-end forwarding delay and the average data loss for the three access mechanisms. As earlier, the opportunistic access mechanism reduces both the forwarding delay and the data loss.

Another network performance parameter, fairness, is used to evaluate the performance of the proposed access mechanism, as shown in Fig. 23. Fairness is calculated in terms of Jain's fairness index [11]. The fairness index value 1 indicates maximum fairness. The figure shows that the fairness index value for the basic access and the four-way access is considerably less compared to the proposed opportunistic access. The packet drops due to hidden nodes result in severe unfairness in the basic access scenario. In the four way access, the R-exposed nodes does not reply back with the CTS packets, which results in high back-off value for the RTS transmitters. This causes unfairness in the network. The problem of unfairness becomes severe as network load increases. The opportunistic access solves both the hidden nodes and exposed nodes, resulting in fairness improvement.

V. Conclusion

This paper shows the severity of the concealed nodes problem in case of the high speed wireless mesh network using theoretical modeling. The analysis shows that the hidden nodes cause severe data losses, and the exposed nodes under-utilize the network capacity by reducing the spatial reuse opportunities. Based on the 802.11n frame aggregation and BACK capabilities, this paper proposes an opportunistic access protocol over the four-way access paradigm to defend the concealed node problems. The effectiveness of the proposed opportunistic scheme is analyzed by results from a practical high speed indoor mesh testbed analysis.

References

[1] A. Tsertou and D. I. Laurenson, "Insights into the hidden node problem," in *Proceedings of the 2006 IWCMC*, 2006, pp. 767–772.

[2] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, "MACAW: a media access protocol for wireless lan's," *SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 4, pp. 212–225, Oct. 1994.

[3] J. Yao, T. Xiong, and W. Lou, "Elimination of exposed terminal problem using signature detection," in *Proceedings of the 9th Annual IEEE SECON*, 2012, pp. 398–406.

[4] J. Alonso-Zarate, L. Alonso, G. Kormentzas, R. Tafazolli, and C. Verikoukis, "Throughput analysis of a cooperative ARQ scheme in the presence of hidden and exposed terminals," *Mob. Netw. Appl.*, vol. 17, no. 2, pp. 258–266, Apr. 2012.

[5] R. Bruno, M. Conti, and E. Gregori, "IEEE 802.11 optimal performances: RTS/CTS mechanism vs. basic access," in *Proceedings of the 13th IEEE PIMRC*, vol. 4, 2002, pp. 1747–1751.

[6] I. Tinnirello, S. Choi, and Y. Kim, "Revisit of RTS/CTS exchange in high-speed IEEE 802.11 networks," in *Proceedings of the Sixth IEEE WoWMoM*, 2005, pp. 240–248.

[7] N. Hajlaoui, I. Jabri, and M. Benjemaa, "Experimental study of IEEE 802.11n protocol," in *Proceedings of the seventh ACM WiNTECH*, 2012, pp. 93–94.

[8] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *IEEE J. on S. Areas in Comm.*, vol. 18, no. 3, pp. 535–547, 2000.

[9] RaLink RT3352 series ieee 802.11n routers-on-chip. [Online]. Available: http://www.mediatek.com/_en/01_products/04_pro.php?sn=1006

[10] M. E. M. Campista, P. M. Esposito, I. M. Moraes, L. Costa, O. Duarte, D. G. Passos, C. V. N. de Albuquerque, D. C. M. Saade, and M. G. Rubinstein, "Routing metrics and protocols for wireless mesh networks," *IEEE Network*, vol. 22, no. 1, pp. 6–12, 2008.

[11] R. Jain, A. Durresi, and G. Babic, "Throughput fairness index: An explanation," Tech. rep., Department of CIS, The Ohio State University, Tech. Rep., 1999.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TWC.2015.2480398, IEEE Transactions on Wireless Communications

Alleviating Hidden and Exposed Nodes in High-Throughput Wireless Mesh Networks

Sandip Chakraborty, *Member, IEEE*, Sukumar Nandi, *Senior Member, IEEE*, Subhrendu Chattopadhyay

***Abstract*—This paper proposes an opportunistic approach to mitigate the hidden and exposed node problem in a high-throughput mesh network, by exploiting the frame aggregation and block acknowledgment (BACK) capabilities of IEEE 802.11n/ac wireless networking standard. Hidden nodes significantly drop down the throughput of a wireless mesh network by increasing data loss due to collision, whereas exposed nodes cause under-utilization of the achievable network capacity. The problem becomes worse in IEEE 802.11n/ac supported high-throughput mesh networks, due to the large physical layer frame size and prolonged channel reservation from frame aggregation. The proposed approach uses the standard 'Carrier Sense Multiple Access' (CSMA) technology along with an 'Opportunistic Collision Avoidance' (OCA) method that blocks the communication for hidden nodes and opportunistically allows exposed nodes to communicate with the peers. The performance of the proposed CSMA/OCA mechanism for high throughput mesh networks is studied using the results from an IEEE 802.11n+s wireless mesh networking testbed, and the scalability of the scheme has been analyzed using simulation results.**

***Keywords*-IEEE 802.11n; Frame Aggregation; Hidden and Exposed Nodes; Spatial Reuse**

I. Introduction

Wireless Mesh Networking [1] technology provides a key milestone to the next generation backbone access network, where a 'mesh' of wireless routers supports the backbone connectivity to the end-users through multi-hop communications. To assist commercial, enterprise and community mesh backbone technology, IEEE 802.11s [2] is gaining significant attention among the developers for its compatibility with commercially successful IEEE 802.11 wireless networking standard. The IEEE 802.11s amendment to the wireless networking standard contributes to the medium access control (MAC) specifications for the wireless mesh access, that can operate along with any physical layer technology supported by the IEEE 802.11 task groups. The recent developments over IEEE 802.11 technology enhances wireless communications for high data rate supports, up to 600 Mbps with IEEE 802.11n, and up to 6.77 Gbps with IEEE 802.11ac [3]. Therefore the mesh networking standard along with high data rate support has exorbitant potentials to provide high-throughput backbone network connectivity to the end users, that effectively directs towards the design of 'wireless world' [4].

The IEEE 802.11 coordination function for wireless channel access relies on 'Listen before Talk' which is known as 'Carrier Sense Multiple Access with Collision Avoidance' (CSMA/CA). The clear channel assessment (CCA) in CSMA/CA uses two listen or carrier sensing (CS) mechanism - physical CS (PCS) and virtual CS (VCS). Extensive studies on IEEE 802.11 CCA technologies have revealed that PCS fails to detect nodes which are beyond the CS range (known as hidden nodes), whereas VCS reduces spatial reuse by blocking the communications which are not susceptible to interference (known as exposed nodes) [5]. The performance impacts of hidden and exposed nodes in a network is characterized by the CS range and the interference range. In PCS, a node can initiate a new communication only if all other nodes in its CS range are inactive. However, the communication becomes successful only if the receiver is not within the interference range of any other active nodes in the network. As CCA is performed at the transmitter, whereas the receiver is susceptible to interference, there is a high probability of hidden nodes in a mesh network. VCS [6] is proposed to solve the hidden node problem through a handshaking between the transmitter and the receiver before the actual communication takes place, through a pair of messages termed as 'Request to Send' (RTS) and 'Clear to Send' (CTS). The transmitter sends a RTS message only if the CCA reports no active nodes in the CS range of the transmitter. The receiver replies back with a CTS message only if there are no active nodes within the CS range of the receiver. On overhearing the RTS and the CTS messages, other nodes within the CS range of transmitter and receiver block their communication for the time duration mentioned within the RTS and the CTS messages. As a consequence, VCS introduces the problem of exposed nodes which are outside the receiver's interference range, but within the CS range of the transmitter. Though the exposed nodes are harmless for an ongoing communication, they are blocked as a result of VCS and therefore cause under-utilization of the effective network capacity.

The performance issues experienced by hidden and exposed nodes in a multi-hop and mesh network have been well studied in the literature. Most of the existing approaches design mechanisms to eliminate either the hidden nodes or the exposed nodes, such as busy tone signaling [7], use of separate control channels [8], adaptation in CS threshold and temporary disabling the CS mechanism [9] etc. However, none of these approaches can eliminate hidden and exposed nodes simultaneously. In fact, existing research [9] has shown that it

Sandip Chakraborty is with the Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, Kharagpur 721302, INDIA (e-mail: sandipc@cse.iitkgp.ernet.in)

Sukumar Nandi and Subhrendu Chattopadhyay are with the Department Computer Science and Engineering, Indian Institute of Technology Guwahati, Guwahati 781039. INDIA (e-mail: {sukumar,subhrendu}@iitg.ernet.in)

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TWC.2015.2480398, IEEE Transactions on Wireless Communications

is very difficult to eliminate hidden and exposed nodes simultaneously, as these are two complementary problems. Further in general for a network with low to moderate traffic load, VCS may show negative impact (considerably less performance than PCS) because of RTS-CTS signaling overhead and large number of exposed nodes [10]. As a consequence, researchers have developed adaptive PCS mechanisms [11] where the CS range is dynamically tuned to reduce the number of hidden nodes in a network. In adaptive PCS, transmit power is tuned in such a way so that the CS range becomes almost equal to the interference range. In this way, the hidden nodes which are within the interference range, but outside the CS range of the transmitter, can be eliminated. Nevertheless, these mechanisms of dynamic CS range can not eliminate the hidden nodes which are within the CS range (or the interference range, assuming both are tuned to be same) of the receiver, but outside the CS range of the transmitter. In [12], the authors have proposed an access point (AP) cooperation method, where the APs share some common information among themselves to detect the hidden and exposed nodes. However, their scheme requires synchronization among the APs which is hard to achieve in a mesh network.

Our earlier paper [13] presents a theoretical model for the performance of IEEE 802.11n supported mesh networks in the presence of hidden and exposed terminals. The high throughput wireless networking technologies over IEEE 802.11n/ac support MAC layer frame aggregation and block acknowledgment (BACK) strategies to improve channel access performance by minimizing the MAC layer overhead [3]. The analysis reveals that the MAC layer frame aggregation and BACK are more susceptible to performance losses in the presence of hidden and exposed terminals. IEEE 802.11n supports a frame aggregation limit of 64 KB. Further in IEEE 802.11n/ac, high data rate support is obtained at the cost of higher and collocated channel losses during interference [14] due to channel bonding [1]. Though individual MAC protocol data units (MPDU) in an aggregated frame (called A-MPDU) can be recovered from random losses, a collocated loss due to collision from hidden terminals destroys a consecutive numbers of MPDUs. Further a BACK loss due to collision from hidden terminals may result in as large as 64 KB data loss, as the complete A-MPDU is required to be retransferred for a BACK loss. Similarly in VCS, a single exposed node unnecessary defers its transmission for a long time. For example with 1500 bytes MAC frame size and 32 Mbps data rate with IEEE 802.11g, the exposed node has to wait for approximately 0.04 ms, whereas with 64 KB MAC aggregated frame size and 600 Mbps data rate with IEEE 802.11n, an exposed node has to wait for approximately 0.1 ms, 2.5 times more. Possibility of an exposed nodes in a mesh network with VCS is very high. As shown in [13], reducing hidden nodes through a tunable CS threshold only is not sufficient to have a reasonable performance benefit in high throughput mesh networks, whereas pure VCS may show negative impact if number of exposed nodes crosses a limit. Considering the extortionate data loss due to hidden nodes in a high throughput mesh network with PCS, and possibility of capacity under-utilization due to exposed nodes in VCS, a scheme should be designed that reduces hidden nodes as much as possible, whereas allows communication opportunities to the exposed nodes.

This paper shows the effect of hidden and exposed terminals over the performance of high throughput mesh networks, using results from a testbed. An improved channel access mechanism is designed in this paper on the top of VCS, utilizing the frame aggregation and BACK strategies in high throughput wireless standards. The proposed mechanism, called CSMA with Opportunistic CA (CSMA/OCA) characterizes the exposed nodes based on their collision properties, and classifies them in transmitter side exposed (T-Exposed, the nodes within the CS range of the transmitter) and receiver side exposed (R-Exposed, the nodes within the CS range of the receiver) nodes. The CSMA mechanism is augmented to allow transmission from the T-exposed nodes which are outside the CS range of the receiver, and the R-exposed nodes which are outside the CS range of the transmitter. This augmentation in CSMA mechanism allows communication for the exposed nodes those do not result in a collision during MPDU communication. However, collision is still possible between MPDUs from one node, and control frames (BACK, RTS or CTS) from another. An opportunistic collision avoidance (OCA) mechanism is proposed in this paper, where the transmit power levels of control frames are determined in an adaptive and coordinated way, such that data-control collision and control packet loss can be avoided as much as possible. The performance of the proposed scheme is analyzed through the results obtained from a practical IEEE 802.11n mesh networking testbed. The scalability of the proposed scheme has been analyzed using simulation results.

II. Effect of Hidden and Exposed Nodes in a High Speed Mesh Network: PCS vs VCS

A number of works, such as [10] and the references therein, have shown that VCS may employ negative impact on channel access overhead due to signaling overhead and exposed nodes. Tinnirello *et al* [16] have shown both analytically and by simulation results that the situation may become worse in case of high speed networks, as the assumption of short transmission time for control frames does not hold for high speed networks. In this section, we report a series of results from a practical IEEE 802.11n+s testbed to analyze the impact of PCS and VCS over high speed multi-hop and mesh wireless networks.

A. Experimental Setups

In these experiments, we use IEEE 802.11n supported Ralink (presently MediaTek) RT-3352 router-on-chip [17] as the core of the wireless routers. The RT-3352 router-on-chip combines 802.11n draft compliant $2T2R$ MAC along with BBP/PA/RF MIMO, a high performance 400MHz

[1]In IEEE 802.11n, two 20 MHz channels are combined to obtain a 40 MHz channel, whereas in IEEE 802.11ac, four 20 MHz channels are combined to use a single 80 MHz channel. It is well known, that wider channels are more susceptible to random as well as collocated channel losses from interference [15].

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TWC.2015.2480398, IEEE Transactions on Wireless Communications

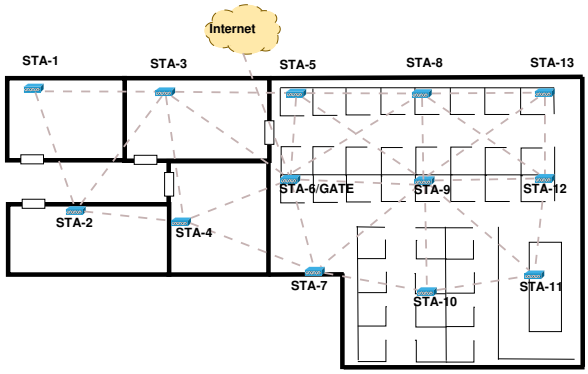


Fig. 1. High Throughput Mesh Testbed and Connectivity Layout

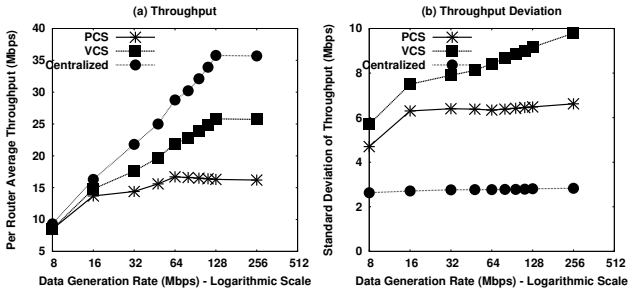


Fig. 2. Comparison among PCS, VCS and Centralized

MIPS24KEc CPU core, a Gigabit Ethernet MAC, 5-ports integrated 10/100 Ethernet Swtich/PHY, 64MB of SDRAM and 32MB of Flash. This chip can support up to 300 Mbps data rate with maximum transmit power of $16dBm$. The routers are configured with Linux kernel version 3.12 with open80211s support [18] which provides open source implementation of IEEE 802.11s mesh networking protocol stack.

B. Experimental Results

Fig. 1 shows the deployment and connectivity layout for a 13 node IEEE 802.11n+s mesh testbed, where 12 nodes act as mesh routers, and one node (STA-6) connects the mesh backbone to the Internet. All 13 nodes are able to act as an access point (AP) to the client nodes to provide backbone mesh connectivity. The dotted lines in the figure shows connectivity among the mesh routers based on the maximum transmit power and receiver sensitivity setup (maximum transmit power 16 dBm, receiver sensitivity -81 dBm). In the testbed, we have used two types of data flows generated through `iperf` tool - the upload traffic flow and the download traffic flow. The upload traffic flows are from mesh routers to the mesh gate (STA-6) and the download traffic flows are from mesh gates (STA-6) to mesh routers. Out of the total traffic flows, 60% data are from download traffic and 40% data are for upload traffic. The performance is evaluated by varying the data generation rate which in turn changes the traffic load of the network. The performance of PCS and VCS is evaluated with a comparison to an centralized scheduling scenario [19] which is generated based on a centralized formulation. The centralized formulation uses a spatial time division multiple access (STDMA) based scheduling, where exposed nodes are allowed to communicate while hidden nodes are blocked explicitly. This scheduling mechanism gives a theoretical upper bound along with protocol overhead, for IEEE 802.11 access scheduling. Therefore, we consider this scheme as the benchmark for performance comparison in this paper, and show how close our proposed distributed scheme can perform compared to the centralized solution. During the design of the centralized scheduling, we have assumed interference range equals to the CS range, as interference beyond the CS range (such as, additive interference) is difficult to design in a decentralized system [20]. In the testbed evaluation, the A-MPDU aggregation level is considered to be 64 KB. The effect of dynamic rate selection is also evaluated, and the performance data is reported later in this paper.

Fig. 2(a) depicts the channel access strategies for three mechanisms - PCS, VCS and the centralized access [19], with respect to average per router throughput. The x-axis shows the mean data generation rate, where every router generates data based on a log-normal distribution along the mean with standard deviation as 2 Mbps. The data generation rate does not include the forwarding traffic. Fig. 2(b) compares the three mechanisms with respect to the standard deviation of routers' throughput. While the average throughput shows the network performance, the standard deviation of throughput indicates effect of hidden and exposed terminals over the throughput performance. As all the nodes transmit traffic towards STA-6, the effect of hidden and exposed terminals is less for the nodes which are in one-hop away from STA-6. Fig. 2(a) shows that throughput loss becomes substantial at high traffic load, when the data generation rate is more than 10 Mbps. Though VCS improves performance at high traffic load by reducing hidden terminals, the throughput is still significantly less compared to the centralized scenario. At the same time, Fig. 2(b) indicates that the throughput deviation is very high for PCS and VCS at high traffic load. From extensive analysis of router traces, we have found that packet loss in PCS is very high for the nodes which experience hidden terminals. However, throughput variation for VCS is higher than PCS, that indicates severe network under-utilization at the presence of exposed nodes. In a high speed mesh network with high traffic load, a single exposed node at every transmission opportunity[2] may lead to network under-utilization equivalent to 64 KB data transmission. As a consequence, the throughput deviation inflates at high traffic load.

The above results indicate that IEEE 802.11 PCS and VCS fail to achieve even half of the centralized throughput in a high data rate wireless mesh network. Therefore, the standard CS mechanisms are required to be revisited for implementing high throughput mesh networks based on IEEE 802.11 standard. The target is to block hidden nodes to avoid data losses due to collision, and to allow transmissions to the exposed nodes for effective utilization of available network capacity. However, enabling communication to all the exposed nodes may create additional problems in a mesh network, as discussed in the

[2]A transmission opportunity indicates the time required to transmit an A-MPDU once a node gains access to the channel

next section.

III. Exposed Nodes: Spatial Transmission and Data-Control Collisions

Modern hardware supports with dynamic power adaptation and capture enabled devices [21] can reduce interference from hidden nodes by tuning CS range to make it equivalent to the interference range, although cannot eliminate them completely. Whereas receiver coordination similar to VCS is necessary to detect the nodes which are outside the CS range of the transmitter, however within the CS range (or interference range) of the receiver. In this design, we consider a network, where CS range is tuned to make it equal to the interference range through some existing mechanism similar to [22], and our target is to further reduce the hidden nodes which are beyond the scope of detection through tuning the CS range, while allowing communication to the exposed nodes as much as possible. The proposed CSMA/OCA protocol works on the top of VCS to improve the performance of high throughput mesh networks by reducing number of hidden nodes and opportunistically allowing communication to the exposed nodes.

As a protocol level simplification, CSMA/OCA considers only the nodes which are within the CS range, and are detectable through either PCS or VCS. The proposed protocol overlooks interference from outside the CS range (additive interference) which is not pre-detectable through PCS or VCS and difficult to capture in a real-time environment. For this purpose, this paper differentiates the term 'collision' and 'interference' as follows: collision is considered for the nodes which are within the CS range and are detectable, whereas interference is considered for the nodes which are outside the CS range, but results in data loss due to additive interference.

During PCS, the nodes which are within the CS (or interference) range of the receiver, but are outside the CS range of the transmitter can be a potential hidden node, as any transmission from it remains undetectable to the transmitter though it causes collision at the receiver. The VCS solves this problem by blocking all the nodes which are within the CS range of both the transmitter and the receiver. As a consequence exposed nodes may result network under-utilization as follows:

1) The nodes which are within the CS range of a transmitter, say T_1, but are outside the CS (or interference) range of a receiver, get blocked by overhearing the RTS message from T_1. These nodes can be a potential transmitter for some other nodes in the network, as they are not within the interference range of a receiver. These nodes are termed as *T-Exposed* nodes.
2) The nodes which are within the CS (or interference) range of the receiver, say R_1, however are outside the CS range of a transmitter, get blocked by overhearing the CTS message from R_1. Though these nodes can not be a transmitter, nevertheless they can be a potential receiver for some other nodes outside the CS range of R_1. These nodes are termed as *R-Exposed* nodes.

Fig. 3-Case 1 shows transmission and collision scenarios for T-Exposed nodes. The dark and light circles denote the transmission range and the CS range respectively. $C \rightarrow D$ communication is feasible as the data communications do not overlap at the receiver, and therefore collision does not exist. However, VCS blocks such communication resulting in node C to be an T-Exposed node. Although, allowing communication to such node may result in data-control collision, as shown in Fig. 3-Case 1(b). The BACK from node B may collide with the data from node C. Similar situation may arise for the CTS from node D. Therefore, special care has to be taken while allowing transmissions to the T-Exposed nodes. Fig. 3-Case 2 shows transmission and collision scenarios for R-Exposed nodes. The communications $A \rightarrow B$ and $C \rightarrow D$ does not result in a collision. However in VCS, node D gets blocked on overhearing the CTS frame from node B. Similar to the earlier scenario, allowing communication to R-Exposed nodes may result in a data-control collision, where one MPDU or two consecutive MPDUs may get lost in an A-MPDU. Based on these collision scenarios, an opportunistic access mechanism is proposed in this paper, as discussed in the next section.

IV. Opportunistic Access: Design and Analysis

The discussions till now have shown that the exposed nodes result in severe under-utilization of spatial reuse in high data rate mesh networks. Though VCS suffers from the signaling overheads, with a little modification in the CTS frame structure, the exposed nodes can be detected within the CS range of the transmitter and the receiver. However, allowing transmissions to all the exposed nodes may result in data-control collision, as shown in Fig. 3. Therefore, an opportunistic access mechanism is introduced in this section to deal with this problem. The opportunistic access mechanism utilizes VCS, where RTS and CTS messages are used for the detection of hidden nodes, as well as allowing transmission to the exposed nodes in an opportunistic way such that data-control collision can be minimized. The OCA mechanism uses an adaptive and coordinated transmit power calculation mechanism, where the control frames are transmitted based on a predefined power level such that data-control collisions can be avoided as much as possible. The detailed design of the opportunistic access mechanism is discussed in the following subsections.

A. Carrier Sensing: Detection of Hidden and Exposed Nodes

The legacy VCS mechanism uses a table, called the *network allocation vector* (NAV), to maintain the transmission blockage on overhearing the RTS and the CTS frames. The RTS and the CTS frames contain a duration field (DU) which indicates the time duration for the channel reservation. On overhearing the RTS and the CTS frames, every node sets its NAV for the time mentioned in the DU field. In the proposed augmentation of VCS, different NAVs are maintained for every overheard RTS and CTS frame, to detect the exposed nodes and to avoid the data-control collision. Let, RTS_{act} and CTS_{act} denote the sets of nodes from which a node has received the RTS or CTS frames, respectively, and $\mathcal{F}.DST$ denote the destination address associated with the frame $\mathcal{F}$. Further assume that $\mathcal{N}_i$ denotes the set of nodes which are in the CS range of node

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TWC.2015.2480398, IEEE Transactions on Wireless Communications

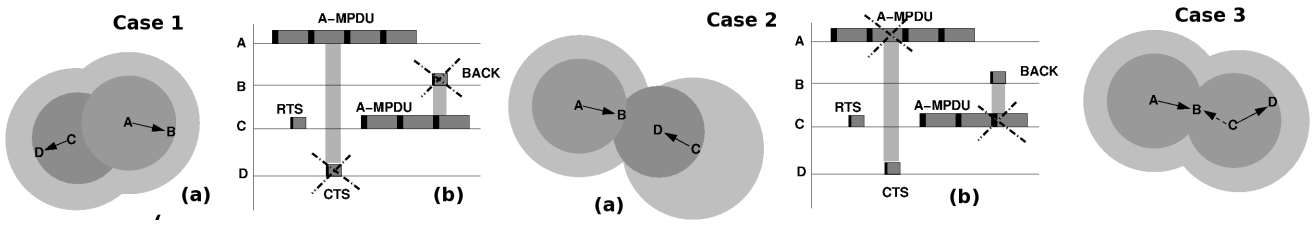


Fig. 3. **Case 1:** Spatial Transmissions and Collision Scenario for T-Exposed Nodes, **Case 2:** Spatial Transmissions and Collision Scenario for R-Exposed Nodes, **Case 3:** Hidden Node Scenario

Algorithm 1 Node S wants to transmit data to node R

1: **if** $(CTS_{act} = NULL) \wedge (\forall RTS_{act}.DST \notin \mathcal{N}_S)$ **then**
2: Calculate η_S.
3: Send <RTS,$\eta_S >$ using P_{max}; /*T-exposed nodes*/
4: **else**
5: Back-Off and retry;
6: **end if**

Algorithm 2 Node R receives RTS from node S

1: **if** $(RTS_{act} = NULL) \wedge (\forall CTS_{act}.DST \notin \mathcal{N}_R)$ **then**
2: Calculate $P_{Tx}^{(R)}, \eta_R$;
3: **if** $P_{Tx}^{(R)} \in \mathcal{P}$ **then**
4: Send $< CTS, \eta_R >$ using $P_{Tx}^{(R)}$; /*R-exposed nodes*/
5: **end if**
6: **end if**
7: $RTS_{act} \leftarrow this.RTS$ /*Append the received RTS in RTS_{act}*/

Algorithm 3 Node S receives CTS from node R

1: **if** $(CTS_{act} = NULL) \wedge (\forall RTS_{act}.DST \notin \mathcal{N}_S)$ **then**
2: Calculate $P_{Tx}^{(S)}$.
3: Send DATA using $P_{Tx}^{(S)}$;
4: **else**
5: Back-Off and retry;
6: **end if**
7: $CTS_{act} \leftarrow this.CTS$ /*Append the received CTS in CTS_{act}*/

i. This set can be populated using the standard IEEE 802.11 beaconing procedure.

We also include an additional 1 byte field, called background noise level (denoted as η_i for node i), at every control frame header. The background noise level indicates the total incident power on the receiver, before the communication starts. The background noise level is measured in terms of received signal strength, expressed in dBm, before transmitting a control packet. Assume that node B is an intended receiver. Therefore following the VCS mechanism, node B needs to transmit a CTS packet after it receives a RTS successfully. The received signal strength at node B is the indication of channel noise near the receiver. This information is propagated to the transmitter, in the form of background noise level, so that it can adjust its power level accordingly. This background noise level value is used for OCA along with the VCS mechanism, as discussed in the subsequent subsection.

The decision controls for hidden and exposed node detection are explained through Algorithm 1, Algorithm 2 and Algorithm 3. According to these algorithms, every data and control frames are transmitted using a predefined power level. The actual power level to transmit the data and control frames are calculated based on the OCA mechanism, as discussed later. In these algorithms, $P_{Tx}^{(i)}$ denotes the transmit power for the data or control packet to be transmitted, and P_{max} denotes the maximum available power level. $\mathcal{P}$ denotes the set of available power levels according to the physical layer modulation and coding technology.

The control statements in these algorithms define whether a node should initiate a communication based on the VCS mechanism. As mentioned earlier, whenever a node overhears a RTS or CTS control packet, it appends or updates the RTS_{act} and CTS_{act} sets accordingly. The lifetime of an entry in these sets is equal to the duration field mentioned in the corresponding RTS or CTS control packets, that denotes how much time a channel is going to be reserved by the corresponding node. Furthermore, it can be noted that a RTS transmission does not always indicate a data communication. It may happen that the sender fails to receive the CTS packet from the intended receiver due to a busy channel or interference. Therefore, after an entry is appended or updated in the RTS_{act} set, a node senses the channel after a timeout interval[3]. If the channel is found to be idle, the corresponding entry from RTS_{act} is deleted. These three algorithms can correctly identify the hidden nodes as well as both types of exposed nodes, as described in the following subsections.

1) Detection of Hidden Nodes: The hidden node scenario can occur for two cases - (i) a transmitter node is within the CS range of another receiver node, and (ii) a receiver node is within the CS range of another transmitter node. For the first scenario, considering Fig. 3-Case 3, let $A \rightarrow B$ communication starts first. Therefore, node C can overhear the CTS frame from node B, which is included in the CTS_{act} set. According to Algorithm 1, the condition is evaluated to be false, as $CTS_{act} \neq NULL$, and node C starts the back-off without initiating the communication.

For the second scenario, let us assume that $C \rightarrow D$ communication starts first. Node B is outside the CS range of node D, but within the CS range of node C. In this scenario node B acts as a receiver, and therefore, should not initiate a communication. Node B can overhear the RTS from node C, and includes it in the RTS_{act} set. On receiving the RTS frame from node A, node B executes Algorithm 2. However, the condition is evaluated to be false as $RTS_{act} \neq NULL$.

[3]In our implementation, this timeout is kept equal to the duration of $3 \times SIFS + T_{CTS}$, where SIFS is the short inter-frame space duration and T_{CTS} is the duration of a CTS frame in time unit (TU).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TWC.2015.2480398, IEEE Transactions on Wireless Communications

Therefore it does not replies back with the CTS, and the communication is deferred avoiding possible collision due to hidden terminals.

2) Spatial Reuse by T-exposed Nodes: Considering Fig. 3-Case 1, let $A \rightarrow B$ communication starts first. Node C within the CS range of node A wants to initiate a transmission with node D, which is outside the CS range of node A. As node C is outside the CS range of node B, it does not receive any CTS. Further, $RTS_A.DST \neq C$. Therefore, the condition in Algorithm 1 is evaluated to be true, and node C forwards the RTS to node D. Node D is outside the CS range of both the nodes A and B. Therefore, the condition of Algorithm 2 is also evaluated to be true, and D replies back with the CTS, resulting in a communication initialization at node C following Algorithm 3.

3) Spatial Reuse by R-exposed Nodes: Considering Fig. 3-Case 2, let $A \rightarrow B$ communication starts first. As node D is outside the CS range of node A, it does not overhear the RTS. Therefore $RTS_{act} = NULL$. Assume, node D receives an RTS from node C. As node D is within the CS range of node B, it overhears the CTS from node B. However, $CTS_B.DST = A \notin \mathcal{N}_D$. Therefore following Algorithm 2, node D replies back with the CTS, resulting in the communication initialization at node C, as shown in Algorithm 3.

B. Opportunistic Collision Avoidance

As discussed earlier, allowing communications to the exposed nodes may result in data-control collision. The T-exposed nodes may result in control frame loss, whereas, the R-exposed nodes may affect data frames. To avoid such losses, we employ the dynamic power adaptation feature available with the commodity wireless routers. The modern commodity router hardwares support a set of transmit power levels for every physical modulation and coding schemes, that can be tuned by the MAC layer driver module. A higher transmit power improves the possibility of correct reception and decoding of a frame, however increases the interference range due to that communication. A frame can be received and decoded correctly if the received signal to interference and noise ratio (SINR) for that frame is beyond a certain threshold, called the *capture threshold*. For a communication $S \rightarrow R$ between two nodes S and R, this can be expressed as;

$$SINR_R = \frac{P_{tx}^{(S)} G_{SR}}{\eta_R} \geq \mathcal{S}_{thresh} \tag{1}$$

where $\mathcal{S}_{thresh}$ is the capture threshold required to correctly decode the frame, $P_{tx}^{(S)}$ is the transmit power level at node S, G_{SR} is the channel gain between S and R. As mentioned earlier, η_R is the background noise level at the receiver R, that denotes total power at the channel just before node R receives the data frame from S. This implies the total noise power contributed by all other communications except $S \rightarrow R$, near the node R. This noise power interferes with the $S \rightarrow R$ communication.

The OCA mechanism proposed in this paper dynamically adjusts the power level for the frames which may undergo a collision. Otherwise the frames are transmitted using the minimum power level such that the communication can sustain. Overall, the OCA mechanism works as follows:

- CTS and BACK control frames may get lost due to probable collision from a T-Exposed node, as shown in Fig. 3-Case 1. Therefore these control frames are transmitted using a higher power level, whenever the background noise level calculated near the sender is high enough. This power level is decided based on the sender side background noise level information (η) transmitted through the RTS and data frames. If the calculated power level is higher than the available power levels, then the communication is not initiated.
- Data frames may get lost due to probable collision from a R-exposed node, as shown in Fig. 3-Case 2. However, in this scenario, the sender side background noise level is very low, and therefore CTS and BACK control frames are transmitted using the minimum power level such that the communication can sustain. As a consequence, the possibility of data loss due to R-exposed nodes can be reduced.
- The RTS frames are transmitted using maximum available power level (P_{max}), since the maximum power level gives maximum possibility of receiving a frame correctly. The RTS frames are required to be received correctly for further coordination in the OCA mechanism.

The adjustment of the transmit power level for a CTS frame works as follows. Let, node S want to communicate to node R, and accordingly transmit the RTS frame with maximum power level P_{max}. The RTS frame also contains the parameter η_S, the background noise level at node S measured just before transmitting the RTS frame. On reception of the RTS frame, let the device driver of node R reports measured SINR as $SINR_{RTS}$. Then,

$$SINR_{RTS} = \frac{P_{max} G_{SR}}{\eta_R} \tag{2}$$

where η_R is the background noise level at node R measured before the reception of the RTS frame. It can be noted that the device driver of every node i periodically measures the background noise level and stores the value in the variable η_i. The background noise level is also used in the standard PCS and VCS mechanism to check whether the channel is free [23], [16]. For instance, the channel is assumed to be free if η_i is less than the CS threshold.

From Equation (2), node R can calculate the value of the channel gain G_{SR}. Now, let $P_{tx}^{(R)}$ denote the transmit power level of node R, for forwarding the CTS frame to node S. For correct reception and decoding of the CTS frame at node S, Equation (1) needs to be satisfied. Therefore,

$$\frac{P_{tx}^{(R)}\, G_{RS}}{\eta_S} \geq \mathcal{S}_{thresh} \tag{3}$$

Thus,

$$P_{tx}^{(R)} \geq \frac{\mathcal{S}_{thresh}\, \eta_S}{G_{RS}} \tag{4}$$

Since the communication interfaces for all the nodes are assumed to be homogeneous and the VCS mechanism has

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TWC.2015.2480398, IEEE Transactions on Wireless Communications

already blocked the communications from the hidden nodes resulting in no hidden node interference, we can assume $G_{RS} \approx G_{SR}$. So,

$$P_{tx}^{(R)} \geq \frac{\mathcal{S}_{thresh}\ \eta_S\ P_{max}}{SINR_{RTS}\ \eta_R} \tag{5}$$

The CTS frames are transmitted only if $P_{tx}^{(R)} \leq P_{max}$. Otherwise, the communication is not initiated. As discussed earlier, a commodity wireless router supports a set of discrete power levels $\mathcal{P}$. Therefore, $P_{tx}^{(R)}$ is approximated to the next available power level from $\mathcal{P}$, higher than $P_{tx}^{(R)}$.

The transmit power levels for the BACK and data frames are calculated in a similar way. The power adaptation method opportunistically handles both the T-Exposed and R-Exposed nodes. For T-Exposed nodes, η_S becomes high. Therefore CTS and BACK frames are transmitted at a higher power level such that the control frames can be received and decoded correctly. For R-Exposed nodes, η_S is very low. Therefore, CTS and BACK frames are transmitted at lower power levels, reducing the possibility of collision with the data frames from the R-Exposed node.

1) Implementation Issue 1 (Measurement of Background Noise Level and SINR): The proposed OCA scheme relies on the measurement of background noise level and the calculated SINR value. In this work, we use an active measurement technique [24], [25] for measuring the received signal strength before the data communication, which is expressed in terms of background noise level. The MadWiFi like device drivers use a hardware abstraction layer (HAL) that measures the difference between the signal level and the noise level for each packet [26]. This measured value is used as the SINR value in our proposed scheme. Further, the noise level of the channel is measured in each interrupt, after reception of a sequence of packets. In our scheme, the latest channel noise reported by the HAL is used as the value of background noise level, expressed in terms of dBm.

However, it can be noted that both the SINR and the channel noise level fluctuates irregularly. To avoid sudden peaks of SINR and background noise level fluctuations, we use an exponentially weighted moving average (EWMA) mechanism to smooth down the reported SINR and background noise level values. Let $\mathcal{S}_{curr}$ is the reported SINR value by the HAL on receiving a packet, and $\mathcal{S}_{sm}$ is the smoothed average value. $\mathcal{D}_{sm}$ is the deviation in estimated SINR. We use the following iterative equations to estimate the smoothed value of SINR.

$$\mathcal{S}_{sm} = (1-\rho)\mathcal{S}_{curr} + \rho\mathcal{S}_{sm} \tag{6}$$

$$\mathcal{D}_{sm} = (1-\rho)|\mathcal{S}_{sm} - \mathcal{S}_{curr}| + \rho\mathcal{D}_{sm} \tag{7}$$

$$\mathcal{S}_{sm} = \mathcal{S}_{sm} - \mu\mathcal{D}_{sm} \tag{8}$$

Here ρ and μ are design parameters, and chosen to give the effect of previous measurement over the current measurement. In OCA, we use $\rho = 0.1$ and $\mu = 1$. These values are chosen based on experimental experiences that gives good smoothing by avoiding the sudden SINR peaks due irregular fluctuations. The calculated background noise level is also smoothed in a similar way with the smoothing parameters chosen as $\rho = 0.2$ and $\mu = 0.8$.

2) Implementation Issue 2 (Adaptive Modulation and Coding in IEEE 802.11): IEEE 802.11 supports adaptive modulation and coding (AMC) where every node has the flexibility to select the modulation and coding scheme (MCS) based on the channel quality and several other performance factors. The physical data rate and the transmit power depends on the selected MCS level. Therefore in a scheme where dynamic power selection is used to mitigate interference, AMC needs to collaborate with the transmit power selection mechanism to choose the appropriate MCS. It can be noted that for the control frames, the AMC always selects the MCS that provides the lowest data rate, because the lowest data rate sustain for maximum channel noise. Therefore, in the proposed scheme, the transmit power for the control frames is selected from the available power levels supported by the MCS corresponding to the lowest data rate.

The problem is non-trivial for the data frames. Equation (5) gives the minimum transmit power level required for successful decoding of the signal at the receiver. In the implementation of CSMA/OCA, once the minimum transmit power level is decided for a communication session, the AMC selects the MCS level that supports transmit power level greater than $P_{tx}^{(R)}$. The proposed CSMA/OCA and AMC works together as follows:

1) CSMA/OCA selects $P_{tx}^{(R)}$ according to equation (5).
2) AMC considers the MCS levels that support transmit power levels greater than or equals to $P_{tx}^{(R)}$. From this reduced set of supported MCS levels, the best MCS is chosen following the AMC algorithms. Let the selected MCS level be $\mathcal{M}$.
3) $\mathcal{M}$ supports a set of transmit power levels. Let it be $\mathcal{P}_{\mathcal{M}}$. CSMA/OCA selects the minimum power level from $\mathcal{P}_{\mathcal{M}}$ which is greater than or equals to $P_{tx}^{(R)}$.

Following this procedure, the proposed CSMA/OCA works together with the standard AMC procedure of IEEE 802.11.

V. Performance Evaluation of CSMA/OCA

The 13 node IEEE 802.11n+s high data rate mesh networking testbed, as shown in Fig. 1 is used to evaluated the performance of the proposed scheme and compare the results with the standard PCS, VCS as well as with the centralized solution as discussed earlier [19]. The performance is also compared with an adaptive PCS mechanism proposed by Park *et al* [27] and the AP coordination mechanism proposed by Nishide *et al* [12]. In [27], the authors have proposed a distributed carrier sense update algorithm (DCUA), that adaptively tunes the transmit power and carrier sense threshold of every node based on a pricing function calculated using the packet error rate. On the contrary, the AP coordination mechanism [12] detects hidden and exposed nodes based on the frame collision probability received through data and control packets.

The proposed CSMA/OCA is implemented as a loadable kernel module (LKM) in the IEEE 802.11s protocol stack for MAC layer channel access. In the implementation, CSMA/OCA is interfaced with the AMC (Minstrel protocol - default AMC for Linux kernel) and the HAL to handle the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TWC.2015.2480398, IEEE Transactions on Wireless Communications

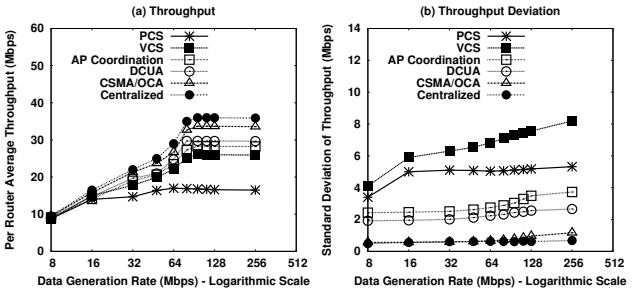


Fig. 4. For Different Channel Access Strategies (a) Throughput, (b) Throughput Deviation

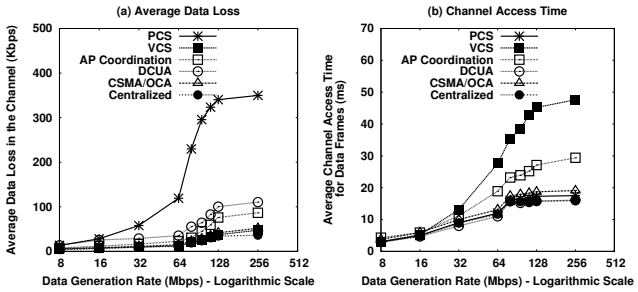


Fig. 5. For Different Channel Access Strategies (a) Channel Data Loss, (b) Average Channel Access Delay for Data Frames

issues related to dynamic MCS selection and calculation of SINR along with background noise level. The system setup is kept similar as discussed in Subsection II-A. The AMC uses the IEEE 802.11n supported MCS levels along with the supported transmit power levels and receiver sensitivity as directed by the RT-3352 hardware [17]. For instance, for MCS $0-8$ and with two different channel widths 20 MHz and 40 MHz, the available transmit powers are $\{3, 5, 8, 11, 14\}$ dBm and $\{5, 8, 11, 14, 16\}$ dBm, respectively.

Fig. 4(a) depicts different channel access protocols with respect to the average per user throughput. The average per user throughput is calculated as the average data transmitted successfully per second. Every mesh router uses A-MPDU aggregation level set to 64 Kbytes. The figure indicates that the network gets saturated (obtains maximum throughput) for PCS when the traffic load is more than 48 Mbps. After the saturation point, the throughput decreases slowly due to increased contention in the network. On the contrary, VCS and other channel access protocols get saturated near 80 Mbps traffic generation rate. The DCUA and AP coordination mechanisms improve average per router throughput compared to the PCS and VCS mechanisms, though the saturation throughput for these schemes are significantly less compared to the centralized strategy. The proposed CSMA/OCA mechanism attains maximum saturation throughput among all the schemes except the centralized strategy, and the performance is close to the centralized strategy. As indicated earlier, the centralized strategy is not implementation-feasible in a wireless mesh network. Being a decentralized scheme, the CSMA/OCA protocol can be easily implemented with marginal modifications in the existing mac80211 module, whereas the network can attain a performance benefit which is comparable with the centralized strategy.

Fig. 4(b) depicts throughput deviation among all the channel access strategies with respect to the traffic load in the network. As indicated earlier, throughput deviation is maximum for PCS and VCS, since hidden and exposed terminals increase unfairness in the network [9]. In presence of hidden and exposed terminals, performances of some nodes are affected by repeated channel access back-off which increases throughput deviation in the network. AP coordination and DCUA reduces this unfairness by eliminating a considerable number of hidden and exposed terminals. However, both of these mechanisms fail to reduce hidden and exposed terminals uniformly in the network. For instance, DCUA can eliminate only the hidden nodes which are within the maximum CS threshold. Similarly, AP coordination can detect only those hidden and exposed nodes which are within the CS range of the neighboring APs. Further AP coordination significantly elevates the control overhead in the network. The proposed CSMA/OCA can eliminate most of the hidden and exposed terminals with less control overhead.

Fig. 5(a) and Fig. 5(b) show the channel data loss and average channel access delay, respectively, for different channel access strategies. The channel data loss indicates the amount of data which are transmitted by a sender, however lost in the channel (acknowledgement is not received) and subsequently retransmitted by the sender. The channel data loss can result from two reasons - collision from the hidden terminals and data loss due to external and additive interference. As the CS range is tuned to make it equal to the interference range through the use of a high carrier sensing threshold value, data loss due to interference is almost negligible. Therefore, Fig. 5(a) indicates the effect of hidden terminals on the performance of the access strategies. On the other side, channel access delay is calculated as the time difference between the time when a MAC data frame is scheduled for transmission (it is at the head of the interface queue), and the time when it is actually transmitted. Therefore, Fig. 5(b) indicates the effect of exposed terminals over the performance. These two figures show that while channel data loss is maximum and grows exponentially for PCS, VCS results in maximum channel access time. AP coordination also has high channel access time, as the coordination requires significant amount of time. It can be noted that the time for control packet exchanges are included within the channel access time. The proposed CSMA/OCA mechanism reduces both the channel data loss as well as average channel access time, resulting in high throughput performance of the network.

Finally we evaluate the effect of A-MPDU aggregation level over the throughput attained through different channel access strategies, as shown in Fig. 6(a) and Fig. 6(b). The mean data generation rate for this experiment is considered to be 64 Mbps (that is just near the saturation level) for Fig. 6(a), and 32 Mbps (the network is unsaturated) for Fig. 6(b). In both the cases, the data generation varies along the mean data generation rate following a log-normal distribution with variance 16 Mbps. The performance of the centralized strategy

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TWC.2015.2480398, IEEE Transactions on Wireless Communications

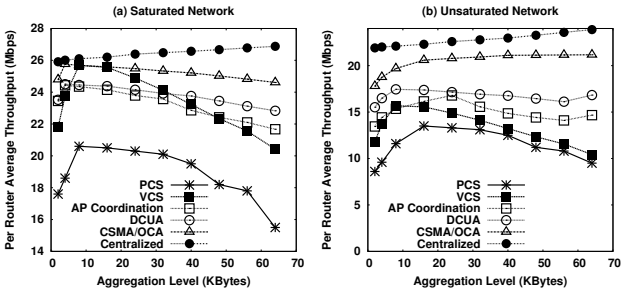


Fig. 6. Effect of A-MPDU Aggregation Level over Per Router Throughput (a) Saturated Network, (b) Unsaturated Network

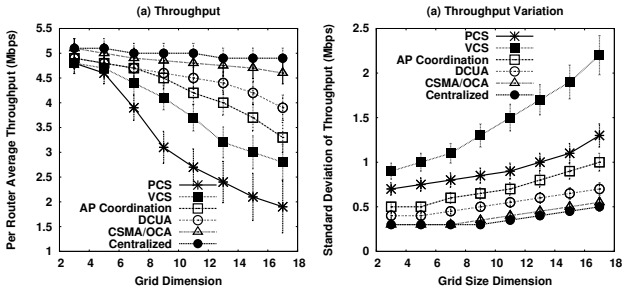


Fig. 7. Simulation Results for Different Channel Access Strategies (a) Throughput, (b) Throughput Deviation

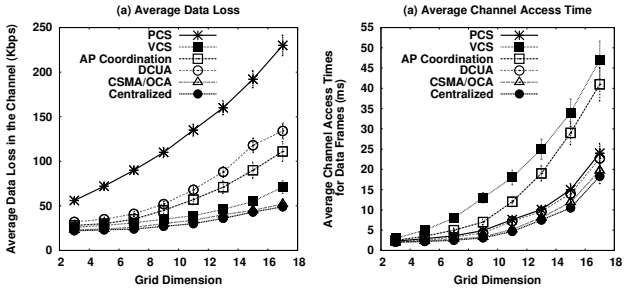


Fig. 8. Simulation Results for For Different Channel Access Strategies (a) Channel Data Loss, (b) Average Channel Access Delay for Data Frames

slowly increases with the increase in data aggregation level, as data aggregation level reduces the physical and MAC control overhead. The performance improvement with respect to the data aggregation level is more prominent in case of a unsaturated network, as shown in Fig. 6(b). With the increase in A-MPDU frame aggregation level, the performance of PCS and VCS first increases, and then drops linearly, and attain a very low throughput compared to the centralized strategy. The performance for PCS and VCS drops after a threshold aggregation level, as the data loss due to collision overshoots the effect of physical and MAC control overheads. With the increase in A-MPDU payload size, the loss due to collision also increases, as discussed earlier. Further VCS shows better performance at low aggregation level compared to AP coordination and DCUA, since AP coordination has significant control overhead, and DCUA can not eliminate hidden terminals completely. The proposed CSMA/OCA scheme performs similar to VCS when aggregation level is low, however, improves the network performance compared to VCS, as well as other schemes, as aggregation level increases. Further, the proposed scheme shows a consistent performance with the increase in the A-MPDU frame aggregation level. These experiments show that the proposed CSMA/OCA results in notable performance improvements in a high throughput wireless mesh network with minimum modifications to the standard mac80211 kernel module at the network protocol stack.

VI. Simulation Results

Although the testbed results give the performance boost for the proposed CSMA/OCA protocol, the experiments are limited with the number of nodes in the testbed. To analyze the scalability of the system, we have simulated the proposed CSMA/CA protocol in Qualnet-5.0.1 network simulator, and compared the results with other related schemes. In Qualnet-5.0.1, we have taken grid topologies of different sizes, with number of nodes in the grid varying from 9 (grid 3×3) to 289 (grid 17×17). Every intermediate node in the grid has 4 neighbors without any cross edges. The mesh gate is placed at the center of the grid. Similar to the testbed scenario, every mesh router has both upload and download traffic. The upload and download traffic are generated using Poisson distribution with the mean and variance as 3 Mbps and 2 Mbps respectively. This indicates a wide variation in traffic distribution across the network. We have executed 10 different scenarios with such random generated traffic model, and plotted the average and standard deviation (90% of the confidence interval) of the results in the graphs. The AMC and power setup for different data rates are kept similar to the testbed setup.

Fig. 7 shows the simulation results for throughput and throughput deviation comparison among different access strategies. In the grasp, grid dimension of n means a grid size of $n \times n$. Following the trend shown by the testbed results as discussed earlier, the simulation results also depict that the proposed CSMA/OCA scheme performs better than others. CSMA/OCA shows a consistent throughput with the increase in grid size.

Fig. 8 shows the average channel data loss and average channel access time for different access strategies. The figure indicates that the proposed CSMA/OCA is scalable to reduce average data loss similar to VCS and average channel access time similar to PCS. The simulation results reveal that the proposed CSMA/OCA mechanism is scalable and provides good performance benefit even in a large network.

VII. Conclusion

This paper presented the severity of the hidden and exposed terminal problem in case of a high throughput wireless mesh network using practical testbed results. The analysis revealed that the hidden terminals cause severe data loss, whereas the exposed terminals under-utilize the network capacity by reducing spatial reuse opportunities. Based on the IEEE 802.11n frame aggregation and BACK capabilities, this paper proposed

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TWC.2015.2480398, IEEE Transactions on Wireless Communications

an opportunistic access protocol over the VCS paradigm to defend the hidden and exposed terminal problems. The effectiveness of the proposed scheme is analyzed through the results from a practical high speed indoor mesh testbed as well as from simulation.

REFERENCES

[1] I. F. Akyildiz and X. Wang, "A survey on wireless mesh networks," *IEEE Communications Magazine*, vol. 43, no. 9, pp. S23–S30, 2005.

[2] G. R. Hiertz, D. Denteneer, S. Max, R. Taori, J. Cardona, L. Berlemann, and B. Walke, "IEEE 802.11s: the WLAN mesh standard," *IEEE Wireless Communications*, vol. 17, no. 1, pp. 104–111, 2010.

[3] E. Perahia and M. X. Gong, "Gigabit wireless LANs: An overview of IEEE 802.11ac and 802.11ad," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 15, no. 3, pp. 23–33, Nov. 2011.

[4] B. Brown and N. Green, Eds., *Wireless World: Social and Interactional Aspects of the Mobile Age*. New York, NY, USA: Springer-Verlag New York, Inc., 2002.

[5] A. Tsertou and D. I. Laurenson, "Insights into the hidden node problem," in *proceedings of the 2006 IWCMC*, 2006, pp. 767–772.

[6] V. Bharghavan, A. Demers, S. Shenker, and L. Zhang, "MACAW: a media access protocol for wireless LAN's," *SIGCOMM Comput. Commun. Rev.*, vol. 24, no. 4, pp. 212–225, Oct. 1994.

[7] J. Monks, V. Bharghaban, and W. M. Hwu, "A power controlled multiple access protocol for wireless packet networks," in *proceedings of IEEE INFOCOM*, May 2007, pp. 219–228.

[8] A. Muqattash and M. Krunz, "Power controlled dual channel (PCDC) medium access protocol for wireless ad hoc networks," in *proceedings of IEEE INFOCOM*, vol. 1, March 2003, pp. 470–480.

[9] L. B. Jiang and S.-C. Liew, "Improving throughput and fairness by reducing exposed and hidden nodes in 802.11 networks," *IEEE Trans. Mobile Computing*, vol. 7, no. 1, pp. 34–49, Jan 2008.

[10] P. M. van de Ven, A. J. Janssen, and J. S. van Leeuwaarden, "Optimal tradeoff between exposed and hidden nodes in large wireless networks," *SIGMETRICS Perform. Eval. Rev.*, vol. 38, no. 1, pp. 179–190, Jun. 2010.

[11] P. van de Ven, A. Janssen, and J. van Leeuwaarden, "Balancing exposed and hidden nodes in linear wireless networks," *IEEE/ACM Transactions on Networking*, vol. 22, pp. 1429 – 1443, October 2014.

[12] K. Nishide, H. Kubo, R. Shinkuma, and T. Takahashi, "Detecting hidden and exposed terminal problems in densely deployed wireless networks," *IEEE Transactions on Wireless Communications*, vol. 11, no. 11, pp. 3841–3849, 2012.

[13] S. Chakraborty, S. Chattopadhyay, S. Chakraborty, and S. Nandi, "Defending concealedness in IEEE 802.11n," in *proceedings of the 6th COMSNETS*, Jan 2014, pp. 1–8.

[14] L. Deek, E. Garcia-Villegas, E. Belding, S. Lee, and K. Almeroth, "Intelligent channel bonding in 802.11n WLANs," *IEEE Transactions on Mobile Computing*, vol. 13, pp. 1536–1233, June 2014.

[15] Texas Instruments, "WLAN channel bonding: Causing greater problems than it solves," Tech. Rep., 2003.

[16] I. Tinnirello, S. Choi, and Y. Kim, "Revisit of RTS/CTS exchange in high-speed IEEE 802.11 networks," in *proceedings of the Sixth IEEE WoWMoM*, 2005, pp. 240–248.

[17] MediaTek RT3352 802.11n 2T2R platform (2.4GHz) - single-band 802.11n with 300mbit/s data rates for Wi-Fi access points and routers. Last visited 21 December, 2014. [Online]. Available: http://www.mediatek.com/en/products/connectivity/wifi/home-network/wifi-ap/rt3352/

[18] Open80211s: Open source implementation of IEEE 802.11s for Linux kernel. Last visited 21 December, 2014. [Online]. Available: www.open80211s.org

[19] G. Brar, D. M. Blough, and P. Santi, "Computationally efficient scheduling with the physical interference model for throughput improvement in wireless mesh networks," in *proceedings of the 12th MobiCOM*, 2006, pp. 2–13.

[20] A. Iyer, C. Rosenberg, and A. Karnik, "What is the right model for wireless channel interference?" *IEEE Transactions on Wireless Communications*, vol. 8, no. 5, pp. 2662–2671, May 2009.

[21] J. Jeong, S. Choi, J. Yoo, S. Lee, and C.-K. Kim, "Physical layer capture aware MAC for WLANs," *Wirel. Netw.*, vol. 19, no. 4, pp. 533–546, May 2013.

[22] T.-S. Kim, H. Lim, and J. C. Hou, "Improving spatial reuse through tuning transmit power, carrier sense threshold, and data rate in multihop wireless networks," in *proceedings of the 12th MobiCOM*, 2006, pp. 366–377.

[23] J. Deng, B. Liang, and P. Varshney, "Tuning the carrier sensing range of IEEE 802.11 MAC," in *proceedings of the IEEE GlobeCOM*, vol. 5, 2004, pp. 2987–2991.

[24] C. Reis, R. Mahajan, M. Rodrig, D. Wetherall, and J. Zahorjan, "Measurement-based models of delivery and interference in static wireless networks," in *Proceedings of the 2006 SIGCOMM*, 2006, pp. 51–62.

[25] W. L. Tan, P. Hu, and M. Portmann, "Experimental evaluation of measurement-based SINR interference models," in *Proceedings of the 2012 IEEE WoWMoM*, June 2012, pp. 1–9.

[26] RSSI in MadWiFi. Last visited 21 December, 2014. [Online]. Available: http://madwifi-project.org/wiki/UserDocs/RSSI

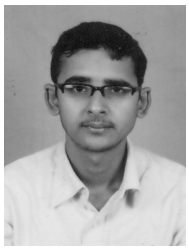
[27] K.-J. Park, L. Kim, and J. C. Hou, "Adaptive physical carrier sense in topology-controlled wireless networks," *IEEE Transactions on Mobile Computing*, vol. 9, no. 1, pp. 87–97, 2010.

Sandip Chakraborty Sandip Chakraborty received the PhD degree from Indian Institute of Technology Guwahati, India in 2014. At present, he is working as an Assistant Professor at Indian Institute of Technology Kharagpur, India. His research interests include wireless networks, mobile computing and distributed computing.

Dr. Chakraborty is a Member of IEEE, IEEE Communications Society and Association for Computing Machinery.

Sukumar Nandi Sukumar Nandi received the PhD degree in Computer Science and Engineering from Indian Institute of Technology Kharagpur, India. He is currently a Senior Professor of computer science and engineering with Indian Institute of Technology, Guwahati, Assam, India. He is coauthored of a book entitled *Theory and Application of Cellular Automata* (IEEE Computer Society). His research interests include traffic engineering, wireless networks, network security and distributed computing.

Dr. Nandi is a Senior Member of IEEE, a Senior Member of Association for Computing Machinery, a Fellow of The Institution of Engineers (India) and a Fellow of The Institution of Electronics and Telecommunication Engineers (India).



Subhrendu Chattopadhyay Subhrendu Chattopadhyay received his B.Tech degree from West Bengal University of Technology and M.Tech degree from Indian Institute of Technology, Guwahati, India. He is currently working towards his PhD degree with Indian Institute of Technology, Guwahati, Assam. He has received a research fellowship from TATA Consultancy Services, India. His current research interests are broadly in the area of computer networking, distributed systems and social network analysis.

Leveraging the Trade-off Between Spatial Reuse and Channel Contention in Wireless Mesh Networks

Subhrendu Chattopadhyay
IIT Guwahati
Guwahati, India 781039
subhrendu@iitg.ernet.in

Sandip Chakraborty
IIT Kharagpur
Kharagpur, India 721302
sandipc@cse.iitkgp.ernet.in

Sukumar Nandi
IIT Guwahati
Guwahati, India 781039
sukumar@iitg.ernet.in

***Abstract*—Performance optimization strategies in wireless mesh networks have witnessed many diverged directions, out of which the recent studies have explored the use of spatial multiplexing through data rate and transmit power adaptation to increase spatial re-usability while minimizing network interference. However, joint data rate and transmit power adaptation shows a clear trade-off, where higher transmit power helps in sustaining high achievable data rates with the cost of increased interference to the neighboring receiver nodes. Such a trade-off results in channel access unfairness among contending flows, where a node with high transmit power gains more performance benefit compared to its neighbors. Unfairness among nodes in a mesh network results in performance drops for the end-to-end flows. In this paper, we formulate a multivariate optimization problem to maximize network utilization and fairness, while minimizing average transmit power for all transmitter nodes. The Pareto optimality nature of the vector optimization has been explored to design a distributed localized heuristic where every node individually decides their scheduling slots and transmit power while keeping fairness as a constraint. The performance of the proposed scheme has been evaluated through simulation, and the comparisons with recent studies reveal that it improves fairness that results approximately $10\% - 40\%$ improvement in end-to-end throughput for different network and traffic scenarios.**

***Keywords*-Fairness; Power adaptation; Rate Control; Scheduling; Joint Optimization; MCCA**

I. Introduction

Wireless mesh network (WMN) is one of the promising technologies for providing cost efficient and robust backbone infrastructure for broadband connection over metropolitan area networks. IEEE 802.11 task group for wireless local area network provides the standard IEEE 802.11s [1], that defines the media access control (MAC) layer channel access and forwarding for multi-hop wireless mesh networks. IEEE 802.11s supports mesh coordination function (MCF) for supporting quality of service (QoS) provisioning in mesh functionalities, which defines MCF controlled channel access (MCCA) for MAC layer scheduling of mesh nodes.

The design and performance of MAC layer protocols for a multi-hop mesh networks significantly depend on several physical layer parameters - like modulation and coding schemes (MCS), transmit powers, channel widths etc, which are further interdependent. Higher MCS levels provide better physical data rate, however require higher transmit power for sustainability. If a transmitter uses higher transmit power level, the data decoding probability at the receiver increases by supporting higher signal to interference and noise ratio (SINR). This can be realized by the following relation between transmit power, physical data rate and SINR [2];

$$SINR = \frac{P_{ij}G_{ij}}{\eta + \sum_{\substack{k \in I \\ k \neq i \\ x \neq j}} G_{kj}P_{kx}} \geq \gamma(r_h) \tag{1}$$

Here P_{ij} and G_{ij} denote the transmit power level and "antenna and channel gain" for a wireless link between transmitter i to receiver j, respectively. I is the set of nodes in the network. η is the ambient noise and $\gamma(r_h)$ is the receive sensitivity for data rate r_h, where r_h corresponds to the physical data rate for MCS level h. If the transmission power is increased, the other receiver nodes in the communication range of the transmitter experience interference, that reduces spatial re-usability of the available channel space. From Eq. (1), it is also evident that the physical data rate has also a direct relationship with the transmit power in forms of receive sensitivity. Receive sensitivity can be defined as the required SINR for correct decoding of a frame transmitted in a particular data rate, that withstands a standard bit error rate (BER) [3]. If SINR of a received frame is greater than the receive sensitivity, then only a node can decode the frame successfully. It has been observed that higher the data rate, greater the receive sensitivity. Notationally for different data rates $r_1 < r_2 < ... < r_m$, $\gamma(r_1) < \gamma(r_2) < ... < \gamma(r_m)$. In summary, the trade-off between data rate and transmit power with respect to spatial re-usability and interference can be captured as follows:

(i) Higher data rates require higher receiver sensitivity, which implies that they can sustain only at higher SINR levels.
(ii) Higher transmit powers provide higher SINR, however in the cost of increased interference and reduced spatial re-usability of the channel.

Therefore, choosing a proper scheduling of transmit power level along with proper MCS level to gain the optimum throughput is a challenge in case of WMN. In a recent study, Li *et.al.* [4] have shown that optimal power control strategy with scheduling even for only two discrete power levels is $\mathcal{NP}$-hard because of the additive/cumulative interference effect on the receiver node.

978-1-4673-9622-6/16/$31.00 ©2016 IEEE

Optimal power control can indirectly improve system performance in terms of throughput. In their work, Cui *et.al.* [5] have suggested use of dynamic programming based solution for controlling radio parameters. Though improvement in network throughput is desirable, it does not ensure improvement in end user throughput. Therefore, increase in fairness at network level is also important as the end users experience better QoS. In this paper, we design a vector optimization problem to simultaneously control data rate, transmit power and MAC layer access parameters, that improves network fairness, spatial re-usability and end user performance. The major contributions of this paper can be summarized as follows.

1) A multi-objective optimization problem is formulated for maximizing network utilization while ensuring fairness. Simultaneously, the optimization targets in minimizing the total transmit power among the contending nodes. The solution of this problem results the assignment of transmit power and MCS along with scheduling parameters such that all links get fair share of the throughput (§III). Mathematical analysis reveals that a Pareto optimal solution for the proposed optimization problem exists (§IV).
2) The problem is centralized in nature and finding a Pareto optimal solution is $\mathcal{NP}$-hard. Therefore a distributed heuristic is proposed by exploring the properties of the centralized problem. Suitable adjustments are made for augmenting the standard MCCA for providing fairness based on the proposed mechanism (§V).
3) Extensive simulations are performed to compare the performance with that of standard MCCA and existing "Distributive Power control Rate adaptation Link scheduling" (DPRL) mechanism [2] to show the improvement in fairness and end user throughput. The results reveal that the proposed scheme results in $10\% - 40\%$ improvements in end-to-end throughput compared to DPRL, in different network and traffic generation scenarios(§VI).

II. Background and Related Works

According to the IEEE 802.11s [1] standard, a WMN consists of two types of nodes called mesh stations (mesh STA) and client stations (client STA). Mesh STAs are the wireless routing entities which form a backbone infrastructure for the client STAs. Some of the mesh STAs act as gateways (mesh gate) which connect the backbone to other network segments or the Internet. Client STAs can connect to this backbone structure for accessing the Internet. A group of mesh STAs collectively form a *"Mesh Basic Service Set" (MBSS)* that uses similar mesh profile among all the STAs.

The standard for IEEE 802.11s MAC describes a set of different channel access protocols. For contention based compulsory channel access mechanism it uses *"distributed coordination function" (DCF)*. The standard also supports for optional *"point coordination function" (PCF)* which is a polling based contention free channel access mechanism. *"Mesh coordination function" (MCF)* is another optional channel access mechanism for ensuring MAC layer quality of service (QoS) for mesh networks. MCF controlled channel access (MCCA) is a *"space-time division multiple access" (STDMA)*[6] based protocol to avoid contention during channel access and to reserve channel resources based on QoS requirements. Each period for MCCA is defined as *"delivery traffic indication message" (DTIM)* interval. DTIM interval is further slotted in MCCA opportunity (MCCAOP) slots. Each reservation consists of three parameters named 1. MCCAOP offset, 2. MCCAOP duration and 3. MCCAOP periodicity. These parameters decide transmission opportunity (TXOP) of a mesh STA. According to the IEEE 802.11s terminologies, a mesh STA that wants to send data, is called a MCCAOP owner, and the corresponding receiver mesh STA is called a MCCAOP responder. The MCCAOP owner and the MCCAOP responder decide TXOPs by exchanging a number of control frames. After the reservation is successful, both the MCCAOP owner and the MCCAOP responder broadcast a MCCA advertisement frame that consists of all the tracked reservations in the sender mesh STA and another parameter, called 'Mesh Access Fraction' (MAF). MAF determines the ratio of the total reserved time by a mesh STA over the total DTIM time period. The maximum reservation period in a DTIM interval for a mesh STA is bounded by MAF limit. It can be noted that the standard scheduling mechanism does not consider physical layer parameters, like transmit power and MCS levels, during scheduling decision.

To address the interdependency between transmit power and MCS, joint design approaches are proposed in the literature. Joint design approach considers the interference relationship between links which are dependent on power level assignment of nodes. In [7], Chen *et.al* have proposed a distributed two phase approach. However their work does not consider the effect of rate adaptation and its dependency with the SINR threshold. As mentioned earlier effective decoding of a received frame depends on mainly two parameters a) SINR and b) MCS. Hedayati *et.al.*[8] modeled the centralized joint integrated power and rate control problem in case of a STDMA network using a mixed integer linear programming (MILP). They have shown that finding a Pareto optimal solution of the MILP problem is $\mathcal{NP}$-hard. They also reduced the problem to maximum independent set problem, and proposed a greedy heuristic solution to the problem. In the subsequent works [2], they proposed a distributed heuristic protocol named DPRL. DPRL protocol uses broadcasting of SINR value. The link with highest SINR margin wins, and then it performs the power and rate calculation based on the channel quality. This process iterates until all the links are scheduled. However, the method suffers from unfairness issues.

The terminology "fair" has different notions in the literature. If achieved throughput of each mesh STA is proportional to it's traffic load then it is called *proportionally fair*. In case of *max-min fair* system, the objective is to maximize the throughput of bottleneck links. It is debatable to use *max-min fairness* [9]. Also being a global property *proportional fairness* is difficult to achieve in a distributed environment. Therefore Mo *et.al.* [9] alternately proposed $(\mathfrak{P}, \alpha)$-Proportional fairness which is

a generalization of *proportional fairness* as well as *max-min fairness*. According to their work let $\mathfrak{P} = \{\mathfrak{P}_{11}, \mathfrak{P}_{12}...\mathfrak{P}_{nn}\}$ and α be all positive numbers. Now a vector of rates $\mathscr{R}^*$ is said to be $(\mathfrak{P}, \alpha)$-Proportionally fair if for any feasible vector $\mathscr{R}_{n\times n}$ under the assumption that, for a n node network with priority of communication being $\mathfrak{P}_{ij}$ between i and j.

$$\sum_{ij} \mathfrak{P}_{ij} \frac{\mathscr{R}_{ij} - \mathscr{R}^*_{ij}}{\mathscr{R}^{*\alpha}_{ij}} \leq 0 \tag{2}$$

The Eq. (3) gives a utility criterion which maximizes overall network throughput under capacity constraints of each links *iff* the rate vector $\mathscr{R}^*$ is $(\mathfrak{P}, \alpha)$-Proportionally fair, as defined in Eq. (2)[1].

$$F_\alpha(\mathscr{R}) = \begin{cases} \mathfrak{P}_{ij} \log(\mathscr{R}) & \alpha = 1 \\ \mathfrak{P}_{ij} \frac{\mathscr{R}^{(1-\alpha)}}{(1-\alpha)} & \text{Otherwise} \end{cases} \tag{3}$$

When $\alpha = 1$, Eq. (3) converges to proportional fairness, and for $\alpha \to \infty$, it reduces to max-min fairness.

However, only transmit power level and MCS selection does not ensure fair allocation of the channel. For ensuring fairness of flows, an explicit strategy needs to be employed so that each mesh STA gets fair share of throughput. It has been studied in [10] that network performance in terms of throughput has a trade-off with fairness of the system. Hence it is challenging to find a strategy which provides fairness for joint power and rate scheduling (JPRS).

III. System Model

IEEE 802.11s MAC layer specification with IEEE 802.11b/g/n physical layer supports multiple frequency channels for contention free channel access mechanism. However many community WMN uses single channel for mesh backbone because static channel assignment for a particular mesh STA limits the neighbor nodes. On the other hand dynamic channel assignment is known to be $\mathcal{NP}$-hard and current heuristics available for channel assignment are costly. So changes in standard are required. The communication channel time is divided into identical channel slots (MCCAOP) of duration σ. Each mesh STA can adjust their transmit power level and MCS from a set of available options. Let power level of mesh STA i for communication $i \to j$ at time t be denoted as $P^{(t)}_{ij}$, where $i \to j$ denotes communication between source i and destination j. Let, r_h be considered as data rate corresponding to h-th MCS. Also consider each mesh STA supports m different such MCSs. R denotes the set of all available data rates. So $R = \{r_1, r_2...r_m\}$ such that $r_1 < r_2 < ... < r_m$. This paper uses the notation $i_x \underset{r_z}{\to} j_y$ for denoting flow i_x to j_y with data rate r_z. In this context r_z and $R(x, y)$ have been used interchangeably throughout this paper.

Let $X^{(T)}_{n\times n\times m}$ denotes the channel reservation matrix of 4-dimension for a network with n mesh STAs which has the following interpretation

$$X^{(t)}_{ijh} = \begin{cases} 1 & \text{If flow i} \to \text{j uses rate } r_h \text{ at time t} \\ 0 & \text{Otherwise} \end{cases}$$

Here T represents number of MCCAOP in a DTIM period. Consider, Tx_{ij} denote number of bits sent by flow $i \to j$ in each DTIM interval. Therefore

$$\begin{aligned} Tx_{ij} &= [DU_{ij} \times Period_{ij} \times r_h \times \sigma] \\ &= \sum_{t=0}^{DTIM} \sum_{h} (X^{(t)}_{ijh} \times r_h \times \sigma) \end{aligned}$$

where $Period_{ij}$ and DU_{ij} denote the MCCAOP periodicity and MCCAOP duration of the corresponding sub-flow respectively. The notation of Tx has been used to refer the vector $[Tx_{i_1j_1}, Tx_{i_2j_2}...Tx_{i_nj_n}]$

It is commonly assumed that "antenna and channel gain" is homogeneous i.e. $G_{ij} \approx G_{ji}$ [11]. The reason behind this assumption is that all wireless routers are equipped with similar type of interfaces. Let receive sensitivity of r_h is denoted by $\gamma(r_h)$.

A feasible transmission scenario $S(t) = \{i_1 \underset{R(1,1)}{\to} j_1, i_2 \underset{R(2,2)}{\to} j_2...i_k \underset{R(k,k)}{\to} j_k\}$ under power allocation vector $P(t) = \{P^{(t)}_{i_1j_1}, P^{(t)}_{i_2j_2}...P^{(t)}_{i_kj_k}\}$ at time t means that the SINR level in each receiver (j_x) must be over threshold ($\gamma(R(x,x))$). Thus Eq. (1) reduces to the following.

$$\frac{G_{i_kj_k} P^{(t)}_{i_kj_k}}{\eta + \sum\limits_{x\neq k} G_{i_xj_k} P^{(t)}_{i_xj_x}} \geq \gamma(R(k,k)) \tag{4}$$

The power assignment for different links are not comparable because the power profile of each mesh STA has to satisfy Eq. (4) individually. Hence cumulative power can not be used as an utility function. Therefore the notion of optimality for power allocation is redefined in this context.

Definition 1: A power vector $P(t)$ is defined to be Pareto optimal for a particular feasible transmission scenario $S(t)$ where any other feasible power assignment $P'(t)$ needs atleast as much power as $P(t)$ for each mesh STA in the same transmission scenario $S(t)$. Mathematically

$$\forall z \in \{1, 2...|P(t)|\} : P^{(t)}_{i_zj_z} \leq P^{'(t)}_{i_zj_z}$$

To achieve Pareto optimal power allocation, this work proposes a multi objective integer program (MOIP) which considers power and rate allocation along with scheduling strategy. However, it has been observed that the link scheduling problem does not consider fairness issues. To achieve the fair share of network throughput, this work considers $(\mathfrak{P}, \alpha)$-proportionally fair scheduling scheme. Using the fairness criteria the objective of the system modifies to achieve Pareto optimal **J**oint **P**ower and **R**ate **S**cheduling which ensures $(\mathfrak{P}, \alpha)$-proportional fairness (Fair-JPRS). The problem tries to achieve the maximum fairness using minimum transmit power level simultaneously.

[1]Here $\log(\mathscr{R}) = \sum\limits_i \log(\mathscr{R}_i)$

IV. Joint Scheduling for Channel Access and Transmit Power Levels: A Centralized Vector Optimization Formulation

In this section, a centralized vector optimization problem is formulated to solve the above mentioned Fair-JPRS problem. The proposed vector optimization problem (VOP) has two aspects.

1) To schedule each station with a power vector that is optimal.
2) Each mesh STA gets a fair share of the resources.

Let $\Gamma(\alpha)$ be an indicator variable such that

$$\Gamma(\alpha) = \begin{cases} 1 & \alpha = 1 \\ 0 & \text{Otherwise} \end{cases} \tag{5}$$

Then according to [12], Eq. (3) can be represented as following

$$F_\alpha(Tx) = \mathfrak{P}_{ij}\left(\Gamma(\alpha)\log(Tx) + (1-\Gamma(\alpha))\frac{Tx^{(1-\alpha)}}{(1-\alpha)}\right) \tag{6}$$

For the sake of notational simplicity let us also consider

$$\mathscr{X}_{ij} = \{Tx_{ij}, P_{ij}\} \qquad \text{and} \qquad \mathscr{X} = \{\mathscr{X}_{i,j} | \forall i, \forall j\} \tag{7}$$

In this case $P_{ij} = \{P_{ij}^{(t)} | \forall t \in \{1, 2...T\}\}$ Clearly the objective functions for scheduling and power allocation is dependent upon $\mathscr{X}_{ij}$. Hence two objectives can be represented in terms of $\mathscr{X}_{ij}$ as following.

$$Schedule(\mathscr{X}) = -\sum_{ij}\left(F_\alpha\left(Tx_{ij}\right)\right) \tag{8}$$

$$Power(\mathscr{X}) = \sum_{ij}\sum_{t}\left(P_{ij}^{(t)}\right) \tag{9}$$

As mentioned earlier, Eq. (8) is to ensure $(\mathfrak{P}, \alpha)$-Proportional fairness. Here $\mathfrak{P}_{ij}$ denotes the priority of the sub-flow $i \to j$ which can be determined by the service class of the flow. However required power vector needs to be minimized as well which can be represented as per Eq. (9). The negative sign in Eq. (8) is used to keep homogeneity of the optimization objective (i.e minimization). Further, each flow must follow the SINR constraint,

$$\Phi[X_{ijh}^{(t)} - 1] - G_{ij}P_{ij}^{(t)} + \gamma(r_h)\sum_{fs} G_{fj}P_{fs}^{(t)} + \gamma(r_h)\eta \leq 0 \tag{10}$$

Here Φ is a substantially large constant which is used to eliminate redundant SINR constraints on sub-flows that are not scheduled at t. Also each mesh STA must be scheduled in such a way that each receiver is different. This is called hidden node constraint. Following equation captures this constraint.

$$\sum_{h}\left[\sum_{ij} X_{ijh}^{(t)} + \sum_{jf} X_{jfh}^{(t)}\right] \leq 1 \tag{11}$$

Now the entire Vector Optimization can be expressed as following.

Problem 1 (Vector Optimization Problem):

$$\text{Minimize} \quad \mathscr{Q}(\mathscr{X}) = \{Schedule(\mathscr{X}), Power(\mathscr{X})\} \tag{12a}$$

S.T:

$$0 \leq P_{ij}^{(t)} \leq P_{max} \quad h \in \{1, 2...m\} \quad t \in \{1, 2...DTIM\} \tag{12b}$$

$$\sum_{h}\left[\sum_{ij} X_{ijh}^{(t)} + \sum_{jf} X_{jfh}^{(t)}\right] \leq 1 \tag{12c}$$

$$\Phi[X_{ijh}^{(t)} - 1] - G_{ij}P_{ij}^{(t)} + \gamma(r_h)\sum_{fs} G_{fj}P_{fs}^{(t)} + \gamma(r_h)\eta \leq 0 \tag{12d}$$

Lemma 1: Every solution of Problem 1 yields a feasible transmission scenario at each time slot.

Proof: Let $\{S(t), P(t)\}$ be a feasible solution of the Problem 1 for any arbitrary scenario and time slot t such that $S(t) = \{i_1 \underset{R(1,1)}{\to} j_1, i_2 \underset{R(2,2)}{\to} j_2, ...i_m \underset{R(m,m)}{\to} j_m\}$ and $P(t) = \{P_{i_1j_1}^{(t)}, P_{i_2j_2}^{(t)}, ...P_{i_mj_m}^{(t)}\}$. Eq. (12c) guarantees that all transmitter and receiver are distinct in each time slot and thus in t also. This prevents hidden node scenario. Let $i_k \underset{R(k,k)}{\to} j_k$ be any arbitrary transmission in $S(t)$ with power profile $P_{i_kj_k}^{(t)}$. Hence based on Eq. (12d), Eq (4) for SINR constraint gets satisfied. This proves that $\{S(t), P(t)\}$ generates all feasible transmission scenario for all the time slots. ■

Theorem 1: All optimum solutions of Problem 1 generates a Pareto optimal power vector allocation based on the transmissions scheduled in each time slot.

Proof: Let $\{S^*(t),P^*(t)\}$ be an optimum solution for Problem 1 at any arbitrary time slot t such that $S^*(t) = \{i_1 \underset{R(1,1)}{\to} j_1, i_2 \underset{R(2,2)}{\to} j_2, ...i_m \underset{R(m,m)}{\to} j_m\}$ and $P^*(t) = \{P_{i_1j_1}^{*(t)}, P_{i_2j_2}^{*(t)}, ...P_{i_mj_m}^{*(t)}\}$. Let the Pareto optimal solution be $P'(t)$. To prove the Pareto optimality of $P^*(t)$, it is enough to prove that $P^*(t) = P'(t)$, component wise. Since there exists only m simultaneous transmissions at t hence only m instances of Eq. (12d) are non redundant. These active constraints can be expressed as follows

$$G_{i_kj_k}P_{i_kj_k}^{*(t)} - \gamma(r_k)\sum_{x \neq k} G_{i_xj_k}P_{i_xj_x}^{*(t)} \geq \gamma(R(k,k))\eta \tag{13}$$

Now the set of constrains in Eq. (13) has a non-negative solution such that $P^*(t) \geq P'(t)$ component wise. This statement can be directly proved by using Perron-Frobenius theorem as [13].

Again the optimality of $P^*(t)$ suggests the fact that $P^*(t) \leq P'(t)$. So we can conclude $P^*(t) = P'(t)$. ■

It can be shown that Problem 1 is $\mathcal{NP}$-hard. For that, a simplistic form of the original problem is considered which incorporates the assumption of $\alpha = 0$, fixed power and fixed MCS. Also protocol interference model [14] is considered. If each sub-flow requires only one transmission opportunity then maximum independent set problem can be easily reduced to the scheduling problem for each time slot. As maximum independent set is a known $\mathcal{NP}$-Complete problem thus we can say Fair-JPRS problem is also $\mathcal{NP}$-hard.

Lemma 2: $Schedule(\mathscr{X})$ is differentiable under $\mathscr{X}_{uv}$ and a convex function.

Proof: Differentiating Eq. (6) w.r.t Tx,

$$\frac{\partial F_\alpha(Tx)}{\partial Tx} = \mathfrak{P}_{Tx}\left(\frac{\Gamma(\alpha)}{Tx} + \frac{(1-\Gamma(\alpha))}{Tx^\alpha}\right) \tag{14}$$

Similarly from Eq. (7),

$$\frac{\partial f(\mathscr{X})}{\partial Tx_{ij}} = \begin{bmatrix}\frac{\partial f(\mathscr{X})}{\partial Tx_{ij}} & \frac{\partial f(\mathscr{X})}{\partial P_{ij}}\end{bmatrix}$$

From Eq. (6)- Eq. (14),

$$\begin{aligned}\frac{\partial Schedule(\mathscr{X})}{\partial \mathscr{X}_{uv}} &= \begin{bmatrix}\frac{\partial Schedule(\mathscr{X})}{\partial Tx_{uv}} & \frac{\partial Schedule(\mathscr{X})}{\partial P_{uv}}\end{bmatrix} \\ &= \begin{bmatrix}-\mathfrak{p}_{uv}\left(\frac{\Gamma(\alpha)}{Tx_{uv}} + \frac{(1-\Gamma(\alpha))}{Tx_{uv}{}^\alpha}\right) & 0\end{bmatrix} \end{aligned} \tag{15}$$

So the Hessian matrix can be defined as,

$$\begin{aligned}[H_s] &= \frac{\partial^2 Schedule(\mathscr{X})}{\partial \mathscr{X}_{uv}^2} \\ &= \begin{bmatrix}\frac{\partial^2 Schedule(\mathscr{X})}{\partial Tx_{uv}^2} & \frac{\partial^2 Schedule(\mathscr{X})}{\partial P_{uv}\partial Tx_{uv}} \\ \frac{\partial^2 Schedule(\mathscr{X})}{\partial Tx_{uv}\partial P_{uv}} & \frac{\partial^2 Schedule(\mathscr{X})}{\partial P_{uv}^2}\end{bmatrix} \\ &= \begin{bmatrix}\mathfrak{p}_{uv}\left(\frac{\Gamma(\alpha)}{Tx_{uv}^2} + \frac{(1-\Gamma(\alpha))}{Tx_{uv}{}^{-(\alpha+1)}}\right) & 0 \\ 0 & 0\end{bmatrix}\end{aligned} \tag{16}$$

$Tx_{uv} > 0$ as it signifies the data transmitted in each DTIM interval for each active flow $u \to v$. $Tx_{uv} = 0$ signifies flow $u \to v$ is inactive for a particular DTIM interval. $[H_s]$ is positive semi-definite as it is symmetric and all its diagonal elements are non negative. Therefore $Schedule(\mathscr{X})$ is a convex function. ■

Lemma 3: $Power(\mathscr{X})$ is differentiable under $\mathscr{X}_{uv}$ and is a convex function.

Proof:

$$\frac{\partial Power(\mathscr{X})}{\partial \mathscr{X}_{uv}} = \begin{bmatrix}\frac{\partial Power(\mathscr{X})}{\partial Tx_{uv}} & \frac{\partial Power(\mathscr{X})}{\partial P_{uv}}\end{bmatrix} = \begin{bmatrix}0 & 1\end{bmatrix} \tag{17}$$

Hence the Hessian matrix is,

$$\begin{aligned}[H_p] = \frac{\partial^2 Power(\mathscr{X})}{\partial \mathscr{X}_{uv}^2} &= \begin{bmatrix}\frac{\partial^2 Power(\mathscr{X})}{\partial^2 Tx_{uv}} & \frac{\partial^2 Power(\mathscr{X})}{\partial P_{uv}\partial Tx_{uv}} \\ \frac{\partial^2 Power(\mathscr{X})}{\partial Tx_{uv}\partial P_{uv}} & \frac{\partial^2 Power(\mathscr{X})}{\partial^2 P_{uv}}\end{bmatrix} \\ &= \begin{bmatrix}0 & 0 \\ 0 & 0\end{bmatrix}\end{aligned} \tag{18}$$

Evidently $[H_p]$ is positive semi-definite. Hence it can be concluded that $Power(\mathscr{X})$ is convex function. ■

Lemma 4: For a feasible transmission scenario, constraints in Eq. (12d) is differentiable under $\mathscr{X}_{uv}$ and convex.

Proof:

$$\frac{\partial P_{uv}^{(t)}}{\partial P_{uv}} = \frac{1}{\frac{\partial P_{uv}}{\partial P_{uv}^{(t)}}} = 1 \qquad \frac{\partial X_{ijh}^{(t)}}{\partial Tx_{uv}} = \frac{1}{r_h\sigma}$$

Under feasible transmission scenario system of constraints, Eq. (12d) presents two types of non-redundant constraints.

$$\Phi[X_{ijh}^{(t)} - 1] - G_{ij}P_{ij}^{(t)} + \gamma(r_h)\sum_{fs} G_{fj}P_{fs}^{(t)} + \gamma(r_h)\eta \le 0 \tag{19}$$

$$\begin{aligned}\Phi[X_{kqh}^{(t)} - 1] - G_{kr}P_{kq}^{(t)} + \gamma(r_h)G_{iq}P_{ij} + \gamma(r_h)\sum_{fs\neq ij} G_{fq}P_{fs}^{(t)} \\ + \gamma(r_h)\eta \le 0\end{aligned} \tag{20}$$

Let L.H.S of Eq. (19) and Eq. (20) be denoted as $SINR(\mathscr{X})$ and $SINR'(\mathscr{X})$, respectively. Hence,

$$\frac{\partial SINR(\mathscr{X})}{\partial P_{uv}} = \frac{\partial SINR(\mathscr{X})}{\partial P_{uv}^{(t)}}\frac{\partial P_{uv}^{(t)}}{\partial P_{uv}} = -G_{uv}$$

Similarly,

$$\begin{aligned}\frac{\partial SINR'(\mathscr{X})}{\partial P_{uv}} = \gamma(r_h)G_{uq}, \quad \frac{\partial SINR(\mathscr{X})}{\partial Tx_{uv}} = \frac{\Phi}{r_h\sigma}, \\ \frac{\partial SINR'(\mathscr{X})}{\partial Tx_{uv}} = 0\end{aligned}$$

$$\frac{\partial SINR(\mathscr{X})}{\partial \mathscr{X}_{uv}} = \begin{bmatrix}\frac{\partial SINR(\mathscr{X})}{\partial Tx_{uv}} & \frac{\partial SINR(\mathscr{X})}{\partial P_{uv}}\end{bmatrix} = \begin{bmatrix}\frac{\Phi}{r_h\sigma} & -G_{uv}\end{bmatrix} \tag{21}$$

$$\frac{\partial SINR'(\mathscr{X})}{\partial \mathscr{X}_{uv}} = \begin{bmatrix}\frac{\partial SINR'(\mathscr{X})}{\partial Tx_{uv}} & \frac{\partial SINR'(\mathscr{X})}{\partial P_{uv}}\end{bmatrix} \tag{22}$$

$$= \begin{bmatrix}0 & \gamma(r_h)G_{uq}\end{bmatrix} \tag{23}$$

■

By the use of Lemma 2, Lemma 3 and Lemma 4 it can be said that Problem 1 is itself a multi-objective convex optimization problem. Hence according to [15], $\mathscr{X}^*$ is a Pareto optimum solution of Problem 1 iff Eq. (24)-(26) has non-negative solution $\forall i : \lambda_i$.

$$\begin{aligned}\lambda_1\frac{\partial Schedule(\mathscr{X}^*)}{\partial \mathscr{X}_{uv}} + \lambda_2\frac{\partial Power(\mathscr{X}^*)}{\partial \mathscr{X}_{uv}} + \lambda_3\frac{\partial SINR(\mathscr{X}^*)}{\partial \mathscr{X}_{uv}} \\ + \sum_q \lambda_4 q\frac{\partial SINR'(\mathscr{X}^*)}{\partial \mathscr{X}_{uv}} = \begin{bmatrix}0 & 0\end{bmatrix}\end{aligned} \tag{24}$$

$$\lambda_1 + \lambda_2 + \lambda_3 + \sum_q \lambda_4 q = 1 \tag{25}$$

$$\lambda_i \ge 0 \tag{26}$$

Eq. (15), Eq. (17),Eq. (21),Eq. (22) and Eq. (24) - Eq. (26) reduce to,

$$\lambda_1 \mathfrak{P}_{uv} \left(\frac{\Gamma(\alpha)}{Tx_{uv}} + \frac{1-\Gamma(\alpha)}{Tx_{uv}^{\alpha}} \right) - \lambda_3 \frac{\Phi}{r_h \sigma} = 0 \tag{27}$$

$$\lambda_2 - \lambda_3 G_{uv} + \sum_q \lambda_{4q} \gamma(r_h) G_{uq} = 0 \tag{28}$$

$$\lambda_1 + \lambda_2 + \lambda_3 + \sum_q \lambda_{4q} = 1 \tag{29}$$

For the sake of simplicity consider $\forall q : \lambda_{4q} = \lambda'_4$ and $\sum_q \lambda_{4q} = \lambda_4$ which further simplify Eq. (27)-(29) into the following,

Problem 2:

$$\lambda_1 \mathfrak{P}_{uv} \left(\frac{\Gamma(\alpha)}{Tx_{uv}} + \frac{1-\Gamma(\alpha)}{Tx_{uv}^{\alpha}} \right) = \lambda_3 \frac{\Phi}{r_h \sigma} \tag{30a}$$

$$\lambda_2 + \gamma(r_h) \lambda'_4 \sum_q G_{uq} = \lambda_3 G_{uv} \tag{30b}$$

$$\lambda_1 + \lambda_2 + \lambda_3 + \lambda_4 = 1 \tag{30c}$$

In Problem 2, Tx_{uv}, G_{uv} and the rest of the constants are positive. Hence it can be shown that there exists a value assignment for $\lambda_1, \lambda_2, \lambda_3, \lambda'_4$ such that they are non-negative.

According to the discussion above, there exists a Pareto optimal solution for Problem 1. However as mentioned earlier, Problem 1 is $\mathcal{NP}$-hard and by virtue of reduction, so is Problem 2. Therefore, in the next section a distributed heuristic is proposed for Fair-JPRS problem.

V. Exploring the Pareto Optimality: A Distributed Heuristic for Fair-JPRS

Each mesh STA controls the transmit power and MCS by sensing the channel condition. Based on the interference level of the channel, each mesh STA is granted some slots (TXOP). However, the interference level of the receiver is very hard to know for a particular flow at the time of transmission. At the start of the protocol, each station uses a test signal with P_{max} to serve the purpose of channel condition assessment. A mesh STA v stores SINR of the received frame from another mesh neighbor u in a variable $\mathcal{S}_{uv}$. $\mathcal{S}_{uv}$ is measured in dB and is piggybacked with MCCA advertisement frames. Before sending a frame, u can always use this piggybacked information to get an estimate of the interference in v by using the following equation which is derived from Eq. (4).

$$\sum_q G_{qv} = \frac{1}{P_{max}} \left(\frac{G_{uv} P_{max}}{\mathcal{S}_{uv}} - \eta \right) \tag{31}$$

Considering $\lambda_2 = \lambda'_4$, the following equation can be derived from Eq. (30b).

$$\lambda_3 = \lambda_2 \frac{1 + \gamma(r_h) \sum_q G_{uq}}{G_{uv}} \tag{32}$$

Considering $\eta \mathcal{S}_{uv}$ is negligible we get,

$$\frac{1}{\lambda_3} \approx \frac{1}{\lambda_2} + \frac{\gamma(r_h) G_{uv}}{\lambda_2 \mathcal{S}_{uv}} \tag{33}$$

From Eq. (30a), Eq. (34) can be obtained as follows.

$$Tx_{uv} = \begin{cases} \frac{\lambda_1 \mathfrak{P}_{uv} r_h \sigma}{\lambda_3 \Phi} & \alpha = 1 \\ (\frac{\lambda_1 \mathfrak{P}_{uv} r_h \sigma}{\lambda_3 \Phi})^{(1/\alpha)} & \text{Otherwise} \end{cases} \tag{34}$$

Eq. (31), Eq. (32) and Eq.(34) are used for solving Problem 2.

Each mesh STA v transmits a periodic beacon frame using P_{max} at the start of the DTIM interval. SINR of the received beacon by mesh STA u is denoted by $\mathcal{S}_{uv}$. Each mesh STA broadcasts its $\mathcal{S}_{uv}$ with MCCAOP advertisement request message. This requires additional frame overhead of 2 Bytes. A mesh STA with highest $\mathcal{S}_{uv}$ in it's neighborhood is termed as "winner", and prioritized to reserve the channel. The assigned rate r_h for winner is chosen so that it is highest achievable data rate with receive sensitivity less than or equal to $\mathcal{S}_{uv}$ *i.e.* $\gamma(r_h) \leq \mathcal{S}_{uv} < \gamma(r_{h+1})$.

$$\mathcal{S}_{uv} = \frac{G_{uv} P_{max}}{\mathcal{I}} \tag{35}$$

$$\gamma(r_h) \leq \frac{G_{uv} P_{uv}^{(h)}}{\mathcal{I}} \tag{36}$$

Here $\mathcal{I}$ represents the cumulative interference and ambient noise. Using Eq. (35) and Eq. (36), we can derive,

$$P_{uv}^{(h)} \geq \frac{\gamma(r_h)\ P_{max}}{\mathcal{S}_{uv}} \tag{37}$$

Though for theoretical modeling purpose this work assumes that each mesh STA has $P_{ij}^{(t)} \in [0, P_{max}]$, in practice commodity routers come with fixed discrete power levels. In that particular case, $P_{uv}^{(h)}$ is approximated to the next higher power level available in routers power level set.

Based on the solution of Problem 2, which is a simplified version of Problem 1, MCCAOP owner calculates MCCAOP parameters in the following way.

1) MCCAOP offset is set as the first free slot in the MCCAOP owner's reservation set.
2) MCCAOP Periodicity is used for ensuring short term fairness of the transmissions. To provide each mesh STA with an equal chance to transmit, this value is set as the number of the MCCA enabled mesh neighbors(Δ).
3) If the winner does not have any prior schedule information, it assigns MCCAOP duration with a predefined fixed value Tx_{max}. If the winner already has a schedule, it estimates $\overline{\lambda}$ using Eq. (31)- (33). Based on the estimated $\overline{\lambda}$, it solves Problem 2. MCCAOP duration is set based on calculated $\lceil \frac{Tx_{uv}}{\Delta} \rceil$.

As per standard, MCCAOP parameters, along with MAF of the MCCA owner, are suggested to the MCCAOP responder.

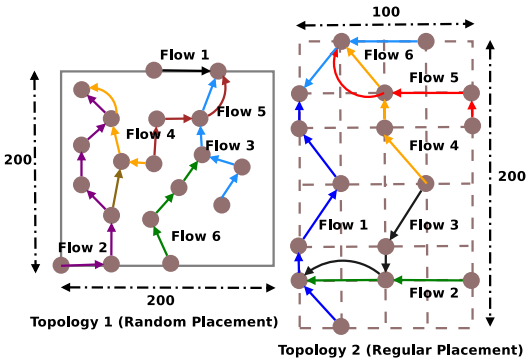


Fig. 1: Simulation Topologies

A MCCAOP responder accepts the schedule if it satisfies the following conditions.

1) MAF is less than MAF limit.
2) All transmission opportunity slots, (TXOP)s do not overlap with the interference period of the responder.
3) All TXOPs must not conflict with the neighborhood MCCAOP periods of the MCCAOP owner.

If the last two conditions are not satisfied, the responder re-calculates TX_{uv} from the received MCCAOP parameters and re-computes MCCAOP parameters for an alternate solution. This alternate MCCAOP parameters are sent back to the MCCAOP owner via MCCAOP Setup reply frame with proper reject code. After receiving successful reservation reply from all MCCAOP responders, MCCAOP owner broadcasts MCCAOP advertisement frame to let the mesh neighbors know about the successful reservation. Once the MCCASCANDURATION is over, each mesh STA starts transmission based on the accepted schedule, data rate and transmit power.

VI. Performance Analysis and Comparison

For determining the performance of the proposed scheme, simulation experiments are performed for two different topologies given in Fig. 1. The different flows are color coded in the two topologies. Simulations are performed in NS-3.18 [16]. Each mesh STA is equipped with single omni-directional antenna. 802.11n physical layer and 802.11s MAC with SDR support are considered for each mesh STA. A single channel is used for the entire duration of the simulation. Channel is time divided into 0.8ms slots. Initial 40 slots are reserved for control and Beacon frames. Rest 1000 slots are used for actual data communication (DTIM). Control frames are transmitted using maximum transmit power and minimum data rate available. For solving Problem 2, α has been chosen uniformly between the range of 3 to 8. More experimental results, performed to realize the effect of α, are omitted due to the space constraint of the paper. Rest of the simulation parameters are given in the Table I. The mesh STAs communicate with the mesh gates through multi-hop communication. Static routing protocol is used for frame forwarding to minimize the influence of routing in each scenario. Each simulation experiment is executed for at-least 10 times. The proposed protocol is compared with the standard MCCA and DPRL [2].

Frame Size	512 B	
Traffic Generation rate	15Mb/s	
MCS	Data Rate	Receive Sensitivity
6.5OFDM	6.5Mbps	-87dBm
26OFDM	26Mbps	-81dBm
39OFDM	39Mbps	-78dBm
54OFDM	54Mbps	-73dBm
Min Power Level	2dbm	
Max Power Level	17dbm	
Power Levels	9	
Slot Time σ	0.80ms	
DTIM	1s	
Slots/DTIM	1000	
Scan Duration	32ms	

TABLE I: Simulation Parameters

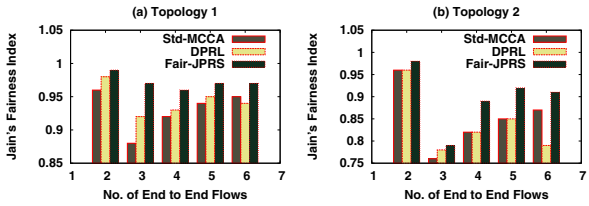


Fig. 2: Effect on Jain's Fairness Index

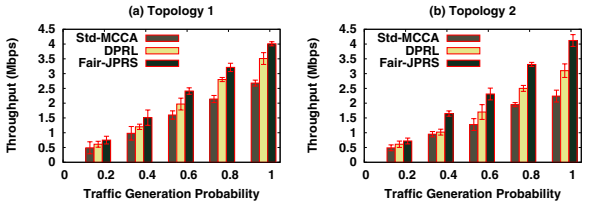


Fig. 3: Effect on End To End Throughput

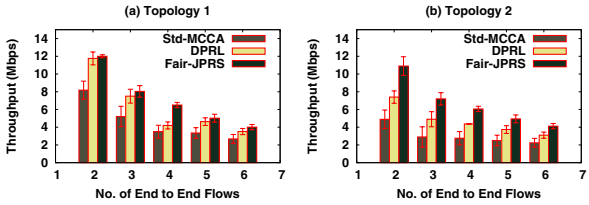


Fig. 4: Effect on End To End Throughput

Simulation results in Fig. 2 show the variation of Jain's fairness index [17], with respect to the number of active end to end flows. In all of the cases, Fair-JPRS gives a better result compared to DPRL and standard MCCA. Fair-JPRS considers neighbor coordinations during transmit power adaptation that helps in improving per node fairness. However in Fig. 2 (b), the graph shows a sharp change for all the protocols when only three flows are active. The reason behind this is, Flow 1 and Flow 3 has a shared bottleneck link and due to this bottleneck link the throughput of the corresponding flow reduces. However Flow 2 does not have any such link. Therefore due to the effect of the specific topology and flow distribution, all of the protocols show lower fairness index.

The improvement in fairness results in significant performance boost in the average end to end throughput. This effect

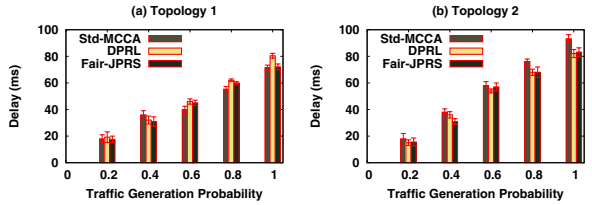


Fig. 5: Effect on End To End Delay

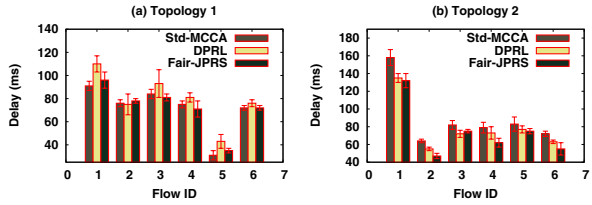


Fig. 6: Effect on End To End Delay

has been captured in Fig. 3[2] and Fig. 4. The confidence factor for average end to end throughput has been calculated based on the standard deviation of all active flows. The nature of the graphs suggest that Fair-JPRS outperforms the rest of two protocols. Per node fairness has direct impact over the end-to-end throughput performance by avoiding the flow stalling at intermediate mesh STAs. As a consequence effect of fairness improvement, Fair-JPRS provides better end-to-end throughput compared to other two schemes. The graphs reveal that depending on the topology and flow distribution scenarios, Fair-JPRS can result in a $10\% - 40\%$ improvement in the end-to-end throughput, compared to DPRL.

It has been studied by Gamal *et.al.* [18] that improvement in throughput might result in the increase in forwarding delay. To analyze the effect of forwarding, we present the variation in delay for all of the protocols. Fig. 5 captures the variation of average delay in case of different data generation probabilities. When all the flows are scheduled, end to end delay of each flow is calculated and shown in Fig. 6. In case of Topology 2, Flow 2 and Flow 3 suffer extra delay than that of DPRL because it receives less bandwidth to ensure fairness for other contending flows. However the difference is very small and for most of the cases the delay remains less than or equal to the end to end delay for DPRL.

VII. Conclusion

Network performance in terms of throughput optimization is a challenge for WMN. In this paper, a fair joint transmit power and rate scheduling problem has been studied in the context of IEEE 802.11s wireless mesh network. For ensuring effective power allocation and throughput fairness, a vector optimization problem is formulated. It has been proved that the problem is a convex optimization problem. The existence of Pareto optimal solution of the problem is proved in this work. However the problem is a $\mathcal{NP}$-hard problem. Therefore a heuristic is also proposed for solving the problem by simplifying the assumptions of the centralized problem. The heuristic proposal is augmented with the IEEE 802.11s standard MCCA. Finally the performance of the scheme is evaluated with the help of simulation results. The simulation reveals that the augmented scheme performs well in terms of end to end throughput. The fairness of proposed protocol shows significant improvement over existing works.

[2]Note: Red marker in the plot, indicates standard deviation

References

[1] "*IEEE* standard for information technology–telecommunications and information exchange between systems local and metropolitan area networks–specific requirements part 11: Wireless LAN medium access control (MAC) and physical layer (PHY) specifications," *IEEE Std 802.11-2012 (Revision of IEEE Std 802.11-2007)*, pp. 1–2793, March 2012.

[2] K. Hedayati and I. Rubin, "A robust distributive approach to adaptive power and adaptive rate link scheduling in wireless mesh networks," *IEEE Transactions on Wireless Communications*, vol. 11, no. 1, pp. 275–283, 2012.

[3] "Cisco aironet 1200 series access point data sheet," http://www.cisco.com/c/en/us/products/collateral/wireless/aironet-1200-access-point.

[4] S. Li, E. Ekici, and N. Shroff, "Power control for AP-based wireless networks under the SINR interference model: Complexity and efficient algorithm development," in *Proceedings of 20th ICCCN*, July 2011, pp. 1–6.

[5] S. Cui, H. Yousefi'zadeh, and X. Gu, "Rate constrained power optimization for STDMA MAC protocols," in *Proceedings of WCNC, 2015*, March 2015, pp. 1129–1134.

[6] K. Amouris, "Space-time division multiple access (STDMA) and coordinated, power-aware MACA for mobile ad hoc networks," in *Proceedings of GLOBECOM 2001*, vol. 5, 2001, pp. 2890–2895 vol.5.

[7] C.-C. Chen and D.-S. Lee, "A joint design of distributed QoS scheduling and power control for wireless networks," in *Proceedings of 25th INFOCOM 2006*. IEEE, 2006, pp. 1–12.

[8] K. Hedayati, I. Rubin, and A. Behzad, "Integrated power controlled rate adaptation and medium access control in wireless mesh networks," *IEEE Transactions on Wireless Communications*, vol. 9, no. 7, pp. 2362–2370, July 2010.

[9] J. Mo and J. Walrand, "Fair end-to-end window-based congestion control," *IEEE/ACM Transactions on Networking*, vol. 8, no. 5, pp. 556–567, 2000.

[10] H. T. Cheng and W. Zhuang, "An optimization framework for balancing throughput and fairness in wireless networks with QoS support," *IEEE Transactions on Wireless Communications*, vol. 7, no. 2, pp. 584–593, 2008.

[11] T. ElBatt and A. Ephremides, "Joint scheduling and power control for wireless ad hoc networks," *IEEE Transactions on Wireless communications,*, vol. 3, no. 1, pp. 74–85, 2004.

[12] J. Jeong, S. Choi, J. Yoo, S. Lee, and C.-k. Kim, "Physical layer capture aware MAC for WLANs," *Wireless networks*, vol. 19, no. 4, pp. 533–546, 2013.

[13] D. Mitra, "An Asynchronous Distributed algorithm for Power Control in Cellular Radio Systems," in *Proceedings of 4th WINLAB Workshop*, Rutgers Univeristy, New Brunswich, NJ, 1993.

[14] Y. Shi, Y. T. Hou, J. Liu, and S. Kompella, "How to correctly use the protocol interference model for multi-hop wireless networks," in *Proceedings of the tenth ACM international symposium on Mobile ad hoc networking and computing*. ACM, 2009, pp. 239–248.

[15] A. Ben-Israel, A. Ben-Tal, and A. Charnes, "Necessary and sufficient conditions for a pareto optimum in convex programming," *Econometrica: Journal of the Econometric Society*, pp. 811–820, 1977.

[16] "Ns-3.18 - nsnam," http://www.nsnam.org/wiki/Ns-3.18.

[17] R. Jain, D.-M. Chiu, and W. R. Hawe, *A quantitative measure of fairness and discrimination for resource allocation in shared computer system*, 1984.

[18] A. Gamal, J. Mammen, B. Prabhakar, and D. Shah, "Throughput-delay trade-off in wireless networks," in *Proceedings of 23rd INFOCOM 2004*, vol. 1, March 2004, pp. –475.

ES2: Managing Link Level Parameters for Elevating Data Rate and Stability in High Throughput WLAN

Sandip Chakraborty
Department of Computer Science and Engineering,
Indian Institute of Technology Kharagpur,
Kharagpur, India 721302
Email: sandipc@cse.iitkgp.ernet.in

Subhrendu Chattopadhyay
Department of Computer Science and Engineering,
Indian Institute of Technology Guwahati,
Guwahati, India 781039
Email: subhrendu@iitg.ernet.in

***Abstract*—High Throughput (HT) IEEE 802.11n/ac wireless networks support a large set of configuration parameters, like Multiple Input Multiple Output (MIMO) streaming, modulation and coding scheme, channel bonding, short guard interval, frame aggregation levels etc., that determine its physical data rate. However, all these parameters have an optimal performance region based on the link quality and external interference. Therefore, dynamically tuning the link parameters based on channel condition can significantly boost up the network performance. The major challenge in adapting all these parameters dynamically is that a large feature set need to be enumerated during run-time to find out the optimal configuration, which is a not feasible in real time. Therefore in this paper, we propose an estimation and sampling mechanism to filter out the non-preferable features on the fly, and then apply a learning mechanism to find out the best features dynamically. We apply a Kalman filtering mechanism to figure out the preferable feature sets from all possible feature combinations. A novel metric has been defined, called the *diffESNR*, which is used to select the best features from the sampled feature sets. The proposed scheme, *Estimate-Sample-Select* (*ES2*) is implemented and tested over a mixed wireless testbed using IEEE 802.11n and IEEE 802.11ac HT wireless routers, and the performance is analyzed and compared with other related mechanisms proposed in the literature. The analysis from the testbed shows that *ES2* results in approx 60% performance improvement compared to the standard and other related mechanisms.**

***Keywords*-High Throughput Wireless; link adaptation; IEEE 802.11n; IEEE 802.11ac**

I. Introduction

The invent and wide-spread deployments of high throughput (HT) wireless technologies, like IEEE 802.11n or IEEE 802.11ax [1]–[3], give a breakthrough to the wireless local area networking (WLAN) for commodity and community wireless usage over free bandwidth. The HT wireless technologies support high physical data rates (IEEE 802.11n supports data rates up to 600 Mbps, whereas IEEE 802.11ax supports rate in Gbps) through a number of physical and media access control (MAC) layer advancements; like the inclusion of multiple antenna technology through *Multiple Input Multiple Output* (MIMO), channel bonding and frame aggregation. The MIMO supports spatial division multiplexing (SDM) with space time block coding (STBC) for improving network capacity. The IEEE 802.11n standard supports 2 spatial multiplexed streams. The channel bonding feature provides wider channels of 40 MHz (for IEEE 802.11n) or 80 MHz (for IEEE 802.11ax) by combining multiple narrow channels of 20 MHz together. Frame aggregation combines multiple upper layer frames, or protocol data unit (PDU), to reduce channel access overhead.

Apart from HT wireless specific features, modern wireless routers also provide different modulation and coding schemes (MCS) that support different data rates. The existing studies [4], [5] have revealed that optimal selection of MCS depends on channel and network conditions, and accordingly different rate adaptation mechanisms have been proposed for legacy WLAN networks. However, HT wireless networks combine a large number of feature set combinations based on the link configuration parameters, like number of spatial streams, channel bondings, guard intervals, MCS levels, frame aggregation levels etc. For example, IEEE 802.11n supports 256 possible combinations of feature sets - 4 spatial streams, 2 channel widths (20 MHz and 40 MHz), 2 guard intervals (800 ns and 400 ns), 8 different MCS levels and 2 frame aggregation levels (ON or OFF). A number of recent studies [5]–[14] have shown that dynamic adaptation from these feature sets provides better network performance compared to static assignments. Accordingly, several link adaptation mechanisms have been proposed in the literature, and Minstrel HT [15] has been widely accepted as the default link adaptation mechanism for HT wireless networks.

The existing link adaptation mechanisms for HT wireless networks can be broadly classified into two groups - open loop link adaptation and closed loop link adaptation. Open loop link adaptation [5]–[7] uses some form of sampling to find out the best set of features from the available feature set. However, the feature sets are extensively large for on-line processing and therefore, the existing approaches use some kind of filtering on static assignments of few features. The closed loop approaches [8] rely on receiver feedback which incurs significant overhead and processing delay to the network. Consequently, the link adaptation mechanism for HT wireless network remains an open issue to the research community.

In summary, link adaptation in HT wireless networks faces two challenges - first, the number of features in the feature space is very large for on-line processing and optimization, and second, both capacity (or throughput) and fairness need to be

978-1-4673-9622-6/16/$31.00 ©2016 IEEE

TABLE I
MCS LEVELS AND CORRESPONDING DATA RATES FOR IEEE 802.11N

MCS	Spatial Streams	Modulation Type	Coding Rate	20 MHz, 400ns	20 MHz, 800ns	40 MHz, 400ns	40 MHz, 800ns
4	1	16-QAM	3/4	39	43.3	81	90
7	1	64-QAM	5/6	65	72.2	135	150
11	2	16-QAM	1/2	52	57.8	108	120
15	2	64-QAM	5/6	130	144.4	270	300

adjusted simultaneously. This paper uses a testbed analysis to understand the impact of different link configuration features; like MIMO spatial streams, MCS levels, channel bonding, guard intervals and frame aggregation, over the performance of the HT wireless networks. Based on the analysis, we design a hybrid link adaptation mechanism to select the best set of link features and tune them adaptively based on channel condition. The proposed link adaptation mechanism works in three phases. In the first phase, we use a Kalman filter approach to predict the signal to noise ratio (SNR) at the transmitter based on receiver feedback. In the second phase, the estimated SNR value is used to design a novel metric, called *differentiated estimated SNR* (diffESNR), which is used to sample the set of features for finding out most appropriate link parameters. *diffESNR* considers performance while reserving fairness during the sampling procedure from the link configuration feature set. In the third phase, the conventional rate adaptation mechanism is applied over the sampled set of features to find out the complete link configuration parameters based on channel condition. The proposed link adaptation mechanism, called *Estimate, Sample and Select* (*ES2*), is implemented over a testbed, and the performance is analyzed and compared with other related works available in the literature. The testbed analysis reveals that *ES2* significantly improves goodput and network fairness compared to `Minstrel HT` [15] and SampleLite [6], two other popular dynamic link adaptation mechanisms, while keeps the signaling overhead minimum.

The rest of the paper is organized as follows. Section II analyzed the impact of link configuration parameters over protocol performance through a thorough testbed analysis. The proposed adaptive link management methodology has been discussed in Section III. Section IV analyzes and compares the performance of the proposed mechanism from the testbed analysis. The impact of proposed *ES2* mechanism over a cross technology testbed (IEEE 802.11n and IEEE 802.11ac mixed testbed) has been discussed in Section V. Finally, Section VI concludes the paper.

II. MOTIVATION: IMPACT OF LINK CONFIGURATION PARAMETERS

In this section, we analyze impacts of different link adaptation parameters, namely channel bonding, MCS selection (no of streams, modulation type and coding rates), guard intervals and frame aggregation (aggregation ON/OFF), over the performance of IEEE 802.11n and IEEE 802.11ac wireless networks.

A. Testbed Configuration

We use three different set of testbed configurations to observe the impact of different link parameters over network performance;

1) IEEE 802.11n only testbed
2) IEEE 802.11ac only testbed
3) IEEE 802.11n/ac mixed testbed

*1) **IEEE 802.11n only testbed**:* : The IEEE 802.11n only testbed consists 18 IEEE 802.11n supported wireless nodes - 6 access points (AP) and 12 wireless stations (STA). The network is configured as a basic service set (BSS), where we vary the number of contending nodes in the BSS. Every node is a RaLink (MediaTek) RT-3352 router-on-chip (RoC) that supports IEEE 802.11b/g/n compatibility. The RoC supports $2T2R$ MAC along with BBP/PA/RF MIMO, a high performance 400 MHz MIPS24KEc CPU core, a Gigabit Ethernet MAC, 5-ports integrated $10/100$ Ethernet Switch/PHY, 64 MB of SDRAM and 32 MB of Flash. This chip can support up to MCS 15 with a peak data rate of 300 Mbps in 40 MHz, 400 ns guard interval. The maximum MCS (MCS 15) supports 2 spatial streams, 64-QAM modulation and $5/6$ coding rate. The APs are connected with Gbps backbone network via wired connection. The RoCs are equipped with Linux Kernel version 3.18. We use `iperf` as the tool for network traffic generation.

*2) **IEEE 802.11ac only testbed**:* : The IEEE 802.11ac only testbed consists 8 IEEE 802.11ac supported wireless nodes - two APs and 8 STAs. As earlier, the network is configured as a basic service set (BSS). Every AP in this testbed is an Asus RT-AC3200 IEEE 802.11ac router, and the STAs are IEEE 802.11ac Asus USB-AC56 client boards. The APs are equipped with 3×3 multi-user MIMO (MU-MIMO) that supports a peak data rate of 1300 Mbps at 5 GHz channel with 80 MHz channel bandwidth. These routers support three channel bonding scenarios: 20 MHz, 40 MHz and 80 MHz. The routers and clients are equipped with open-source `asuswrt-merlin` firmware built over Linux Kernel version 3.18.

*3) **IEEE 802.11n/ac mixed testbed**:* : In this scenario, both the IEEE 802.11n and IEEE 802.11ac routers are used, while the IEEE 802.11ac routers are configured in n+ac mixed mode. The analysis from this testbed configuration shows the interaction and interoperability among IEEE 802.11n and IEEE 802.11ac routers.

The testbed configurations are shown in Figure 1. For IEEE 802.11n testbed, we configure two basic service sets (BSS) with 3 APs each, for IEEE 802.11ac testbed, there are two BSSes with 1 AP each, and in the mixed testbed, we have

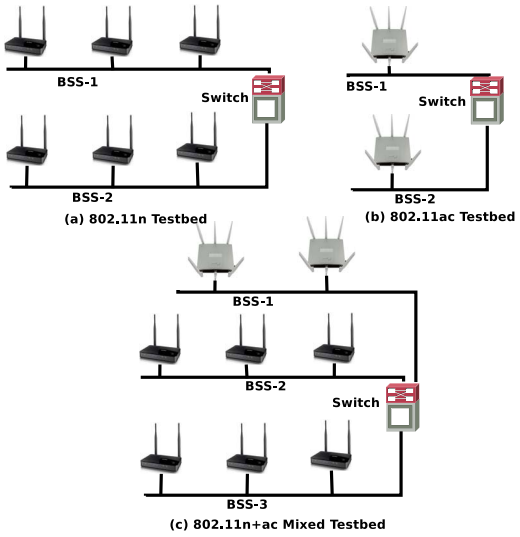


Fig. 1. Testbed Configuration

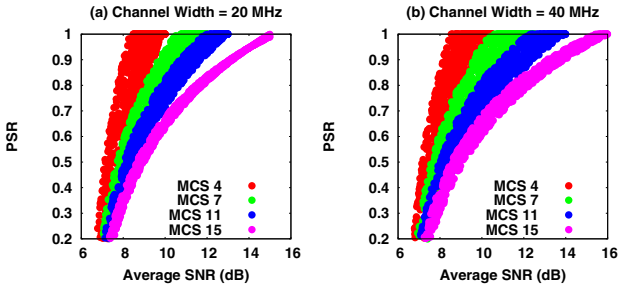


Fig. 2. Impact of MCS Selection over PSR (IEEE 802.11n)

three BSSes for two BSSes with 3 IEEE 802.11n APs and one BSS with 2 IEEE 802.11ac APs. All the BSSes are connected to a common 10 Gbps switch.

B. Protocol and Metric Selection

We use `Minstrel` [16] as the base link adaptation protocol which is widely used in present Linux kernel network modules. `Minstrel` uses packet success rate (PSR) as the metric for rate adaptation. As discussed in several works in the literature [6], [10], [17], the PSR of a wireless link depends on the channel condition that can be measured in terms of signal to noise ratio (SNR). In these sets of experiments, we use SNR and PSR as the measurement metrics.

C. Impact of MCS Selection and MIMO Streaming

To check the impact of MCS selection and MIMO streaming, we fix the channel bonding and short guard interval. All the nodes transmit data using predefined MCS levels. We consider MCS levels 4, 7, 11 and 15. The number of spatial streams, modulation type, coding rates, and data rates corresponding to different {Channel width, Guard Interval} settings for the selected MCS levels have been shown in Table I. We consider 20 MHz, 40 MHz (for IEEE 802.11n) and 40 MHz, 80 MHz (for IEEE 802.11ac) communications at 5 GHz band, for these experiments. The data is collected from all the three testbed configurations (in both pure mode and mixed mode) and the average data points are plotted in the graphs.

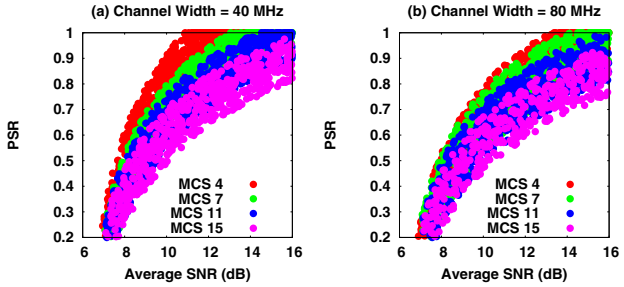


Fig. 3. Impact of MCS Selection over PSR (IEEE 802.11ac)

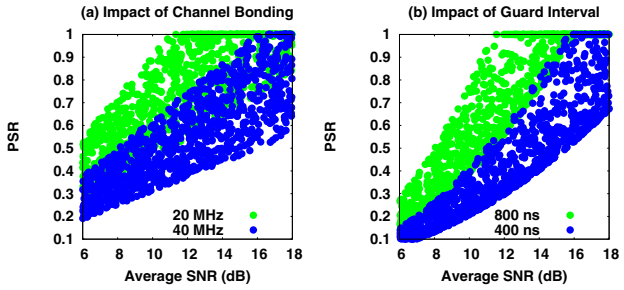


Fig. 4. Impact of Channel Bonding and Guard Intervals (IEEE 802.11n)

Fig. 2 and Fig. 3 show the PSR for different MCS levels with respect to SNR value, for IEEE 802.11n and IEEE 802.11ac, respectively. The figures indicate that low MCS levels can sustain at low SNR region and provide better PSR compared to high MCS values. However high MCS levels provide good PSR at high SNR region. It can be observed further that PSR variation in significantly more in single stream communication compared to double stream communication. Fig. 6(b) reveals that 40 MHz and 80 MHz channels perform better at high SNR region, however it increase PSR variation significantly compared to 20 MHz channel. Such a variation in PSR impacts fairness of the network.

D. Impact of Channel Bonding and Guard Intervals

To analyze the impact of channel bonding, we have generated the SNR-PSR curves for different MCS levels at both 20 MHz and 40 MHz channels for IEEE 802.11n, and additionally 80 MHz channel for IEEE 802.11ac. Due to space constraint, only the graph for MCS 15 is discussed in this paper, as shown in Fig. 4(a) and Fig. 5(a). The figures indicate that 20 MHz performs better in low SNR region. This is also discussed in existing literatures [13], [14] that 40 MHz and 80 MHz wider channels get affected by the external noise and interference. The similar analysis has been done for guard intervals, as shown in Fig. 4(b) and Fig. 5(b). IEEE 802.11n/ac HT wireless

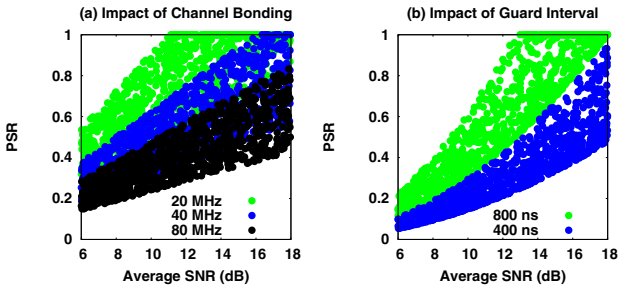


Fig. 5. Impact of Channel Bonding and Guard Intervals (IEEE 802.11ac)

supports short guard interval (400 ns) which is effective for low interference scenario. The figure also indicates that 400 ns performs good at high SNR region.

E. Observations from the Testbed Analysis

The testbed analysis shows that different link parameters, like MIMO streaming, MCS levels, channel bonding, guard interval and frame adaptation together decides the capacity of a HT wireless link. It can be noted that although we have not reported the results for frame aggregation, it has been observed that frame aggregation works better at high SNR region. Therefore the existing works [5], [7], [8], [12] that consider only a subset of these parameters provide a suboptimal solution to the adaptive link management problem. The adaptive link management for HT wireless networks should consider all of these parameters simultaneously to find out the optimal link configuration. However, the major challenge is to enumerate all the possible combinations at runtime to find out the optimal solution which takes infeasible amount of time. Therefore in this paper, we use an estimation and sampling mechanism to filter out the non-preferable configurations before going to find out the optimal solution. The detailed procedure is discussed in the next section.

III. Adaptive Link Management: Estimate, Sample and Select

Based on the observations from the testbed analysis, we design a three phase mechanism for adaptive link management in HT wireless networks. We exploit the acknowledgement (ACK) feedback mechanism, where the receiver piggybacks extra information for coordination with the transmitter. Although several works in the literature use open-loop link adaptation to reduce the transmission overhead, we believe that hybrid link adaptation can perform much better in this scenario because of the complex interdependencies among different parameters along with throughput-fairness trade-off as discussed in the previous section. However, even with a receiver feedback, the problem of adaptive link management is non-trivial because of the accurate estimation of channel quality and the design of switching strategy from one configuration set to another.

The three phase mechanism for adaptive link management in HT wireless networks works as follows:

1) Estimate the SNR at transmitter from the measured received signal strength (RSS) at the receiver,
2) Sample the feature sets based on the estimated SNR thresholds,
3) Select the final data rate from the filtered samples.

The detailed design of the adaptive link management mechanism is given next.

A. Estimation of SNR based on Kalman Filter

Estimation of SNR in HT wireless networks is nontrivial because of two reasons,

(i) The legacy hardware devices measure the received signal strength for each packet arrived at the receiver interface. The noise level significantly depends on parametric settings (number of spatial streams, channel width, short guard interval) of the neighboring nodes, and therefore fluctuates abruptly with respect to time. Therefore simple subtraction of noise margin from the signal strength (as done in many legacy IEEE 802.11 drivers, like MadWiFi and its extensions [18]) may not give a good estimate.

(ii) The transmitter needs to sample from the link feature set, whereas the performance of the link depends on receiver side channel quality, as the receiver gets affected from channel interference.

For the estimation of SNR, we use Kalman Filter [19] based approach to subtract the noise floor estimate from the RSS. Let $\bar{\mathcal{R}}_t$ and $\mathcal{R}_t$ be the estimated RSS and measured RSS (when a packet is received), respectively, at time t. With every ACK packet, the receiver piggybacks the information $\{\mathcal{R}_t,t\}$ from which the transmitter estimates $\bar{\mathcal{R}}_t$. Let the estimated noise be modeled as a Gaussian random variable $\mathcal{N}_t \sim N(0,\mathcal{Q})$ with variance $\mathcal{Q}$. Similarly, the measured noise is captured by another Gaussian random variable $\nu_t \sim N(0,\mathcal{S})$ where $\mathcal{S}$ is its variance. Then the system can be captured using following set of equations;

$$\bar{\mathcal{R}}_t = \bar{\mathcal{R}}_{t-1} + \mathcal{N}_{t-1} \quad (1)$$

$$\mathcal{R}_t = \mathcal{R}_{t-1} + \nu_{t-1} \quad (2)$$

Therefore, we need a model for RSS behavior from eq. (1) and another model from noise estimate from eq. (2). The SNR (dB) can be computed by subtracting the noise floor (dBm) from RSS estimate (dBm).

Let $\bar{\mathcal{R}}_t^-$ and $\bar{\mathcal{R}}_t^+$ be the a-priori and post-priori estimate of the RSS. $\bar{\mathcal{P}}_t^-$ and $\bar{\mathcal{P}}_t^+$ are the a-priori and post-priori estimates of the error variance. Then the *time update equations* [19] for Kalman filter work as follows,

$$\bar{\mathcal{R}}_t^- = \bar{\mathcal{R}}_{t-1}^+ \quad (3)$$

$$\bar{\mathcal{P}}_t^- = \bar{\mathcal{P}}_{t-1}^+ + \mathcal{Q} \quad (4)$$

Similarly, the *filter measurement equations* [19] for the Kalman filter work as follows,

$$\mathcal{K}_t = \bar{\mathcal{P}}_t^-(\bar{\mathcal{P}}_t^- + \int)^{-1} \quad (5)$$

$$\bar{\mathcal{R}}_t^+ = \bar{\mathcal{R}}_t^- + \mathcal{K}_t(\mathcal{R}_t - \bar{\mathcal{R}}_t^-) \quad (6)$$

$$\bar{\mathcal{P}}_t^+ = (1-\mathcal{K}_t)\bar{\mathcal{P}}_t^- \quad (7)$$

Here K_t is the *Kalman gain*. The set of equations given above updates the error variance at every instance of time, and finds out the RSS and the noise floor. It is well understood that Kalman filter minimizes the mean square error at every iteration, and predicts the optimal noise variables for Gaussian distribution.

Therefore in the proposed link quality estimation mechanism, the receiver piggybacks the measures RSS and noise level with the ACK packets, and the transmitter estimates the SNR at every time instance based on the Kalman filter approach. This estimated SNR value is used in the next step to sample the features from the available set of feature set.

B. Sampling of the Feature Set

As discussed earlier, the number of features for adaptive link management in HT wireless networks is significantly large. Therefore, the transmitter needs to sample the possible features from the available feature set, that can be considered in the next step to find out the optimal link setup parameters. For this purpose, we apply a similar method as proposed in [6]. However our scheme use a different metric, termed as *diffESNR*, instead of transmitter side average RSS. The *diffESNR* at time t is defined as the squared difference between the estimated SNR in time t and time $t-1$ multiplied by the SNR at time t. Let $\bar{\mathcal{S}}_t$ be the estimated SNR at time t. Then $\mathrm{diffESNR}_t$ can be defined as;

$$\mathrm{diffESNR}_t = \bar{\mathcal{S}}_t(|\bar{\mathcal{S}}_t^2 - \bar{\mathcal{S}}_{t-1}^2|) \tag{8}$$

The diffESNR parameter considers the current estimated SNR along with the variance in SNR with respect to the time scale. Therefore it intelligently considers both the performance as well as fairness, as the increased variation in SNR may result in unfairness in link adaptation. Whenever the SNR variance is high, diffESNR can select the features which will in turn reduce the performance variation in the link. For example, as shown in Fig. 2 and Fig. 3, 20 MHz works well when SNR variation is high, as it reduces the PSR variation. It has been observed from the testbed analysis that although the SNR value fluctuates abruptly with respect to time and the best link adaptation parameters are not monotonic with respect to SNR, but they follow monotonic behavior with respect to diffESNR. This has been established by following results as discussed next.

In Fig. 6 and Fig. 7, we have shown the effect of diffESNR in the selection of spatial streams and channel bonding. The network and link setup for this experiment is kept similar as discussed in previous section. Fig. 6 and Fig. 7 indicate that a low diffESNR implies single stream link to be more preferable compared to double stream. Intuitively when diffESNR is less, SNR variation gets decreased and also the end-to-end SNR is on the lower side. The single stream link can sustain at low SNR values with small variation. On the other side, high SNR variation can be controlled by double stream links at the presence of high SNR values. Therefore, double stream provides better result in that scenario. The figure indicates that with diffESNR threshold at 20 dB, the communication can be

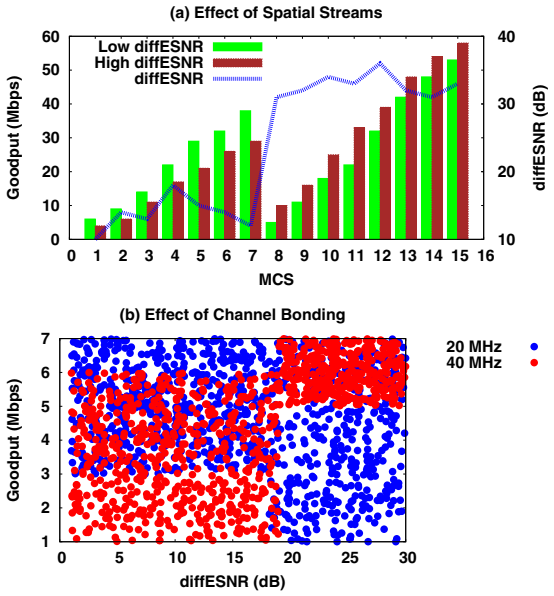


Fig. 6. Effect of diffESNR over Spatial Streams and Channel Bonding (IEEE 802.11n)

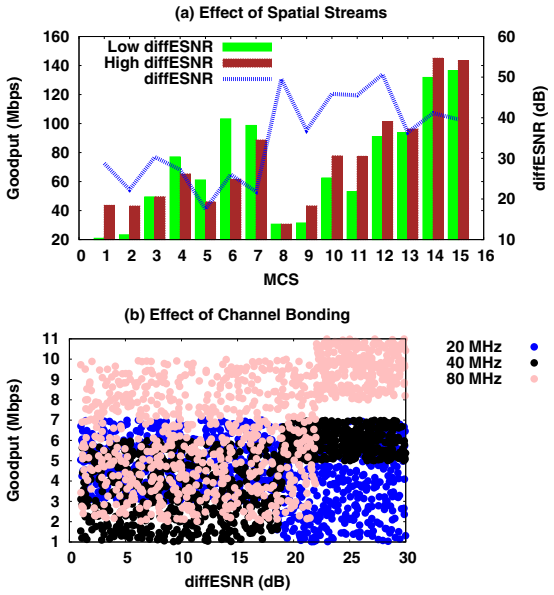


Fig. 7. Effect of diffESNR over Spatial Streams and Channel Bonding (IEEE 802.11ac)

switched from the single stream mode to the double stream mode. Similar observation can be done for the selection of channel bonding. 40 MHz channel requires high SNR value with less SNR fluctuation. To get a fair estimation of the impact of diffESNR over channel bonding, we have used MCS 0 at 40 MHz channel and MCS 1 at 20 MHz channel. MCS 0 at 40 MHz provides 13.5 Mbps whereas, MCS 1 at 20 MHz supports 13 Mbps physical data rate. We compute the average goodput for these two cases, and plot the same in Fig. 6(b) and Fig. 7(b). The figures indicate that at low diffESNR, 20

Algorithm 1 Filtering out Non-Preferable Choices from the Set of Link Features

```
if diffESNR ≤ Threshold(Streams) then
    Stream = Single Stream
else
    Stream = Double Stream
end if
if diffESNR ≤ Threshold(Channel_20) then
    Channel = 20 MHz
else
    if diffESNR ≤ Threshold(Channel_40) then
        Channel = 40 MHz
    else
        Channel = 80 MHz
    end if
end if
if diffESNR ≤ Threshold(GI) then
    GI = 800 ns
else
    GI = 400 ns
end if
if diffESNR ≤ Threshold(Aggregate) then
    Aggregate = OFF
else
    Aggregate = ON
end if
```

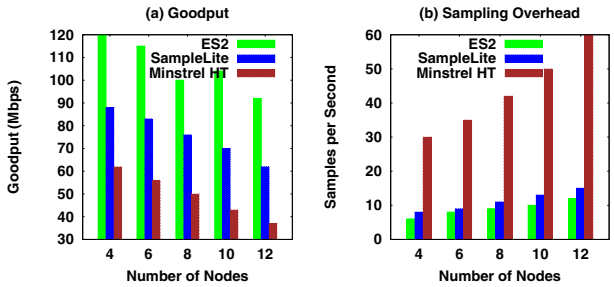


Fig. 8. Static Scenario (IEEE 802.11n)

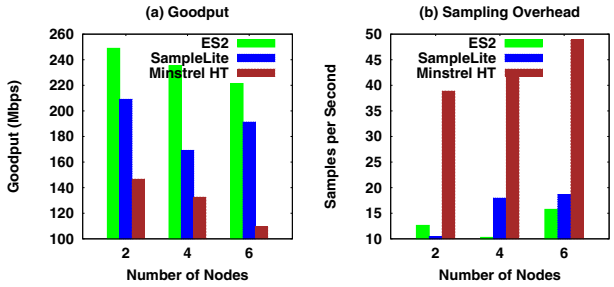


Fig. 9. Static Scenario (IEEE 802.11ac)

MHz gives better result whereas at high diffESNR, 40 MHz outperforms. 80 MHz performs better in further increase of diffESNR value. In this scenario, the switching from 20 MHz to 40 MHz can be triggered at a threshold of 18 dB. Similarly, a transition from 40 MHz to 80 MHz can be triggered with a threshold of 22 dB. Similar observations have been found for guard interval and frame aggregation, however not reported in this paper due to the space constraint.

Based on the above observation, the transmitter defines diffESNR thresholds for the spatial streams, channel bonding, short guard interval and frame aggregation. Based on these thresholds, a simple mechanism is executed for filtering out the non-preferable choices from the set of features, as shown in Algorithm 1. It can be noted that the threshold values are note fixed, but they are adaptive based on the observation of the link performance. Initially the threshold value is fixed to some user defined value based on experimental evidences. We use an interleaved monitoring mechanism, where the transmitter periodically updates one threshold at a time by ± 1 dB and ± 2 dB, transmits a few test HELLO packets and observes the PSR. If there is an improvement in PSR, it updates the corresponding threshold. To bound such signaling overhead, we limit such HELLO packet to 2% of all the data packets transmitted by the corresponding transmitter.

C. Rate Selection

The sampling mechanism reduces the search spaces only to select the appropriate MCS level either from the single stream or from the double stream. The other parameters like channel bonding, guard interval and frame aggregation get fixed during the sampling phase as binary (or distinct n-ary where n is very less: ex. $n = 3$ for channel bonding in IEEE 802.11ac) decision can be made over such parameters. As an example it can be noted that for IEEE 802.11n, the sampling mechanism reduces the search spaces from 256 possibilities to 8 possibilities (every stream supports 8 different MCS levels). At this level, the adaptive link management problem in HT wireless network reduces to conventional rate adaptation problem in wireless networks, where the data rate need to be selected, given that other features are fixed. Any standard rate adaptation mechanism can be used in this context. For this paper, we use `Minstrel` [16], the legacy rate adaptation mechanism for Linux operating system. Once other parameters are fixed, the proposed adaptive link management module executes `Minstrel` rate adaptation to find out the optimal data rate for that selection.

IV. Performance Analysis and Comparison

The proposed adaptive link management, *ES2*, is implemented over the testbed as discussed earlier, as a kernel submodule in mac80211 module. We have also implemented `Minstrel HT` [15] and SampleLite [6], two other link adaptation protocols for HT wireless networks. `Minstrel HT` is the default link adaptation protocol for IEEE 802.11 based HT MAC submodule of Linux kernel, however, it only considers MIMO spatial streams, MCS levels and channel bonding for adaptive link management. SampleLite is the recent and to the best our knowledge, the only link adaptation protocol that considers all the feature sets together. However, SampleLite does a transmitter side RSS calculation for sampling out the non-preferable options.

Two different network scenarios are considered in this paper

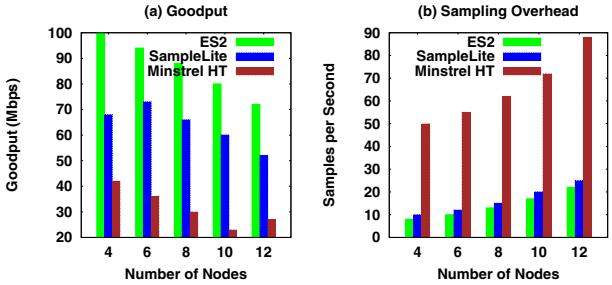


Fig. 10. Mobile Scenario (IEEE 802.11n)

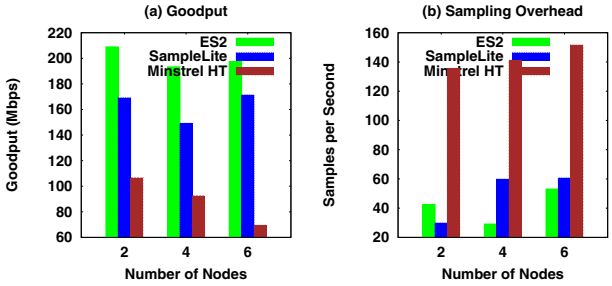


Fig. 11. Mobile Scenario (IEEE 802.11ac)

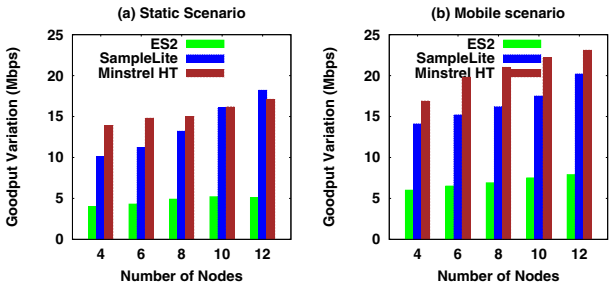


Fig. 12. Fairness: Average Link Goodput Variation (IEEE 802.11n)

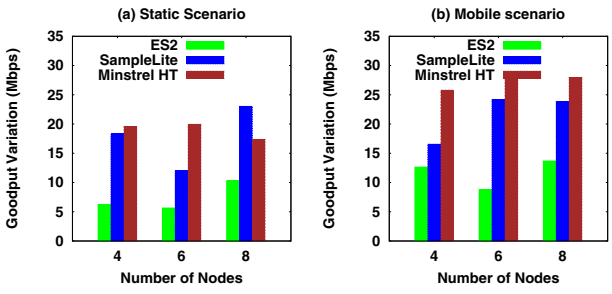


Fig. 13. Fairness: Average Link Goodput Variation (IEEE 802.11ac)

for performance evaluation - one static scenario and one mobile scenario. We consider both IEEE 802.11n and IEEE 802.11ac testbed. In the mobile network scenario, the nodes move at a rate of 1 m/s using random mobility model. We apply both UDP and TCP traffic at the application using

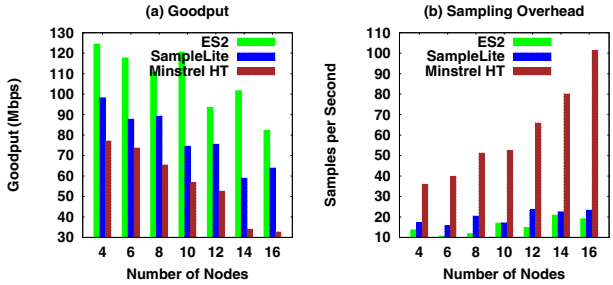


Fig. 14. Static Scenario (Mixed Testbed with Interoperability)

`iperf` tool. The UDP traffic generation rate is 4 Mbps. The UDP and TCP traffic distribution in the network follows UDP:TCP = 30 : 70 ratio. Similarly the upload and download traffic follows upload:download = 40 : 60 ratio.

Fig. 8 and Fig. 9 show the goodput and sampling overhead in static scenario whereas Fig. 10 and Fig. 11 plot the same in the mobile network scenario. It can be noted that the total number of nodes include the HT wireless access points plus the client devices. The figures reveal that there is almost 60% performance improvement compared to SampleLite with a marginal reduction in sampling overhead. Although *ES2* and SampleLite both use similar sampling mechanism, however SampleLite relies on transmitter side RSS which can not predict the link quality correctly. In wireless networks, interference effects are more severe at the receiver. *ES2* uses a novel metric, called diffESNR, that predicts the link quality from receiver side feedback. As a consequence, the proposed *ES2* scheme shows around 60% performance improvement compared to SampleLite.

As discussed earlier, link fairness is another important metric that need to be considered during the link adaptation mechanism. We measure link fairness in terms of average goodput variation per link, that is plotted in Fig. 12 and Fig. 13 for the static and mobile network scenarios. The figures reveal that *ES2* significantly reduces per link goodput variation compared to SampleLite and `Minstrel HT`. The sampling metric, diffESNR, considers the effect of SNR variation which reduces the goodput variation, and in turns improve link fairness.

V. Discussion: *ES2* Operation in a Mixed IEEE 802.11n + IEEE 802.11ac Testbed

Next we show and analyze the results from a mixed network testbed with IEEE 802.11n and IEEE 802.11ac. We configure both the IEEE 802.11n and IEEE 802.11ac access points in 5 GHz channel in fixed channel operating mode. The results are shown in Fig. 14 and Fig. 15, for static and dynamic scenarios, respectively. Similar to the earlier scenario, we can observe a significant goodput improvement for the proposed *ES2* mechanism, with marginal sample overhead. It can be noted that in the mixed mode scenario, we have kept similar set-ups of channel bonding (20 MHz and 40 MHz) and MCS levels (MCS 0 to MCS 15) to make them compatible with each

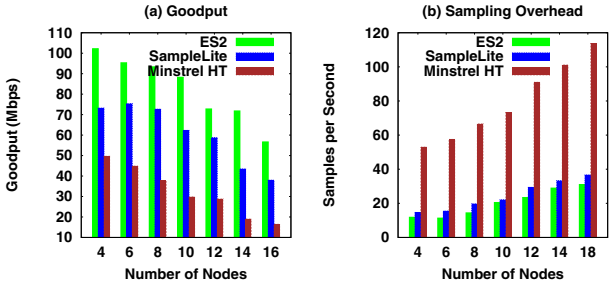


Fig. 15. Mobile Scenario (Mixed Testbed with Interoperability)

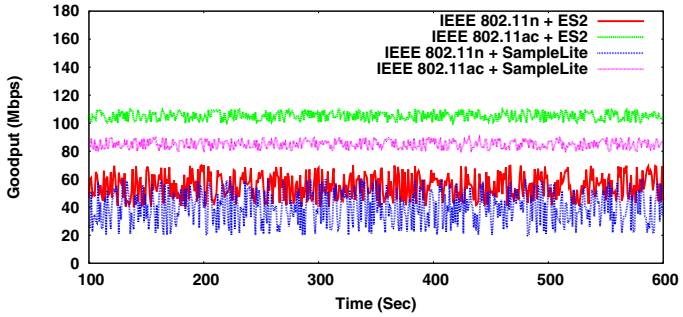


Fig. 16. Performance From IEEE 802.11n + IEEE 802.11ac Mixed Testbed

other. When the access points work in a non-coordinate mode (for example IEEE 802.11ac operates in 80 MHz and IEEE 802.11n works in 40 MHz), IEEE 802.11n nodes fail to detect the set-up for IEEE 802.11ac access points, and therefore the performance for IEEE 802.11n suffers.

We do an experiment over the mixed testbed scenario, as shown in Fig. 1, to analyze and observe the performance of IEEE 802.11n and IEEE 802.11ac access points, when they operate in different configuration set. In this setup, we allow all the available configurations for both the different variants of HT wireless. Fig. 16 shows the average goodput for IEEE 802.11n and IEEE 802.11ac, for *ES2* and SampleLite. The figure indicates that although *ES2* improves the performance for IEEE 802.11ac, many of the times the performance for IEEE 802.11n drops. We analyzed the trace, and observed that when IEEE 802.11ac operates in 80 MHz, which is not supported in IEEE 802.11n, the later fails to coordinate and filter out the non-preferable configurations. As a result, the performance for IEEE 802.11n drops. The heterogeneous mode operation of *ES2* is kept as a future extension of our work as it require a thorough analysis of interoperability.

VI. Conclusion

In this paper, we propose a novel link adaptation mechanism, called *ES2*, that considers different link configuration parameters in HT wireless networks. The proposed mechanism takes into account the effect of MIMO streaming, MCS selection, channel bonding, guard interval and frame aggregation to find out the optimal set of parameter configuration based on link quality assessment. Because of the large volume of feature set, we propose a sampling strategy based on a new metric, called diffESNR. The metric diffESNR is calculated from estimated SNR at the transmitter side, based on receiver feedback, with the help of Kalman filtering approach. The proposed scheme is implemented in a real testbed, and the analysis of the testbed result reveals that *ES2* significantly improves goodput and link fairness while using marginally less control overhead compared to SampleLite, the most recently proposed mechanism for adaptive link management.

References

[1] W. Sun, O. Lee, Y. Shin, S. Kim, C. Yang, H. Kim, and S. Choi, "Wi-Fi could be much more," *IEEE Communications Magazine*, vol. 52, no. 11, pp. 22–29, 2014.

[2] M. X. Gong, B. Hart, and S. Mao, "Advanced wireless LAN technologies: IEEE 802.11ac and beyond," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 18, no. 4, pp. 48–52, Jan. 2015.

[3] D.-J. Deng, K.-C. Chen, and R.-S. Cheng, "IEEE 802.11ax: Next generation wireless local area networks," in *proceedings of QShine*, 2014, pp. 77–82.

[4] S. H. Y. Wong, H. Yang, S. Lu, and V. Bharghavan, "Robust rate adaptation for 802.11 wireless networks," in *Proceedings of the 12th Mobicom*, 2006, pp. 146–157.

[5] I. Pefkianakis, S.-B. Lee, and S. Lu, "Towards MIMO-aware 802.11n rate adaptation," *IEEE/ACM Trans. Netw.*, vol. 21, no. 3, pp. 692–705, 2013.

[6] L. Kriara and M. K. Marina, "SampleLite: A hybrid approach to 802.11n link adaptation," *ACM SIGCOMM Computer Communications Review*, vol. Online First, 2015.

[7] D. Nguyen and J. Garcia-Luna-Aceves, "A practical approach to rate adaptation for multi-antenna systems," in *proceedings of 19th ICNP*, 2011, pp. 331–340.

[8] L. Deek, E. Garcia-Villegas, E. Belding, S.-J. Lee, and K. Almeroth, "Joint rate and channel width adaptation for 802.11 MIMO wireless networks," in *proceedings of 10th SECON*, 2013, pp. 167–175.

[9] L. Kriara, M. K. Marina, and A. Farshad, "Characterization of 802.11n wireless LAN performance via testbed measurements and statistical analysis," in *proceedings of 10th SECON*, 2013, pp. 158–166.

[10] R. Combes, A. Proutiere, D. Yun, J. Ok, and Y. Yi, "Optimal rate sampling in 802.11 systems," in *Proceedings of IEEE INFOCOM*, 2014, pp. 2760–2767.

[11] S. Seytnazarov and Y.-T. Kim, "Cognitive rate adaptation for high throughput ieee 802.11n WLANs," in *proceedings of 15th APNOMS*, 2013, pp. 1–6.

[12] Z. Zhao, F. Zhang, S. Guo, X.-Y. Li, and J. Han, "RainbowRate: MIMO rate adaptation in 802.11n wild links," in *proceedings of IPCCC*, 2014, pp. 1–8.

[13] L. Deek, E. Garcia-Villegas, E. Belding, S.-J. Lee, and K. Almeroth, "The impact of channel bonding on 802.11n network management," in *Proceedings of the Seventh CoNEXT*, 2011, p. 11.

[14] S. Chakraborty and S. Nandi, "Controlling unfairness due to physical layer capture and channel bonding in 802.11n+s wireless mesh networks," in *Proceedings of ICDCN*, 2015, pp. 21:1–21:10.

[15] F. Fietkau and D. Smithies, "minstrel_ht: New rate control module for 802.11n," 2010. [Online]. Available: https://lwn.net/Articles/376765/

[16] W. Yin, P. Hu, J. Indulska, and K. Bialkowski, "Performance of mac80211 rate control mechanisms," in *Proceedings of the 14th ACM MSWiM*, 2011, pp. 427–436.

[17] X. Chen, P. Gangwal, and D. Qiao, "RAM: Rate adaptation in mobile environments," *IEEE Transactions on Mobile Computing*, vol. 11, no. 3, pp. 464–477, 2012.

[18] The MadWifi project, "RSSI in MadWifi," 2015. [Online]. Available: http://madwifi-project.org/wiki/UserDocs/RSSI

[19] M. Senel, K. Chintalapudi, D. Lal, A. Keshavarzian, and E. J. Coyle, "A Kalman filter based link quality estimation scheme for wireless sensor networks," in *Proceedings of IEEE GLOBECOM*, 2007, pp. 875–880.

A time aware method for predicting dull nodes and links in evolving networks for data cleaning

Niladri Sett, Subhrendu Chattopadhyay, Sanasam Ranbir Singh, Sukumar Nandi
Department of Computer Science and Engineering
Indian Institute of Technology, Guwahati
Guwahati, India-781039
Email: {niladri,subhrendu,ranbir,sukumar}@iitg.ernet.in

***Abstract*—Existing studies on evolution of social network largely focus on addition of new nodes and links in the network. However, as network evolves, existing relationships degrade and break down, and some nodes go to hibernation or decide not to participate in any kind of activities in the network where it belongs. Such nodes and links, which we refer as "dull", may affect analysis and prediction tasks in networks. This paper formally defines the problem of predicting dull nodes and links at an early stage, and proposes a novel time aware method to solve it. Pruning of such nodes and links is framed as "network data cleaning" task. As the definitions of dull node and link are non-trivial and subjective, a novel scheme to label such nodes and links is also proposed here. Experimental results on two real network datasets demonstrate that the proposed method accurately predicts potential dull nodes and links. This paper further experimentally validates the need for data cleaning by investigating its effect on the well-known "link prediction" problem.**

Keywords*—Social network analysis, Dull node, Dull link, Time-series, Link prediction.

I. Introduction

Evolution of social network [1], [2] has been receiving substantial attention in the field of social network analysis (SNA). A social *tie* (or a link) is built through social interaction between two *actors* (or nodes); and structure of a social network evolves with addition of new nodes and links in the network. Association or friendship between two actors is maintained by the amount of information that flows between them, which is often quantified using the pattern of interactions they are involved in. Most of the studies on the evolution of social network have concentrated on addition of new links and nodes in it. However, in reality, some links become inactive with time as friendships break down, and some nodes withdraw themselves from the network. Inaction or disappearance of those links and nodes alter the structure of the network. We refer such nodes and links as *dull nodes* and *dull links* respectively. Ignoring dull nodes and links may affect analysis and prediction tasks in the network, because these nodes and links provide spurious information. This paper proposes a novel time aware method to predict dull nodes and links at an early stage as a *preprocessing* task in social networks. We formally define the task of predicting such nodes and links at an early stage by exploiting their historical properties: by observing a time-evolving social network up to time τ, predicting the nodes (and links) which will be declared dull at a future time τ'. The nodes and links, which are idle for long, are labeled as dull. A novel scheme is proposed to fix two time duration thresholds, each for nodes and links of a particular network. If a node or link remains idle longer than its respective threshold, it will be labeled as dull. Removal of predicted dull nodes and links from the network reduces noisy information, and is considered as a *data cleaning* step for dynamic networks.

In recent times, researchers have started exploring temporal dimension in SNA. There are studies [3], [4], [5], [6] on two major SNA tasks namely *link prediction* [7] and *community detection* [8], which consider temporal information as an enhancement to baseline methods. These temporal methods inherently carry properties that can discriminate between an active node (or an active link) and a dull node (or a dull link). However, to the best of our knowledge, none of them has formulated and solved the problem of predicting dull nodes (and dull links), nor considered the time-aware data cleaning for dynamic networks. After pruning the predicted dull nodes and links from the network, the preprocessed network can be used for any SNA task using simple non-temporal baseline methods, which may achieve considerable performance gain along with improvement in running time.

Experiments on two real network datasets demonstrate that the proposed method effectively predicts potential dull nodes and links. We further validate our claim of noise reduction by investigating link prediction [1] performance, subject to data cleaning. We show that, the removal of predicted dull nodes and links results in considerable improvement in performance of state-of-the-art link prediction methods for both datasets.

More specifically, contributions of this paper are as follows.

- It proposes a novel data cleaning method for dynamic networks. This method predicts and removes dull nodes and links, which will eventually become inactive or leave the network in future.
- It also proposes a novel scheme to label the dull nodes and links.
- A case study is presented, which demonstrates the effect of the proposed data cleaning method on state-of-the-art link prediction methods.
- Experiments are performed on two real network datasets, which show that the proposed data cleaning

[1]The link prediction problem in social network is defined as: given the snapshot of a network at time t, predicting the links that will appear at a future time t' [7].

978-1-5090-4470-2/16 $31.00 © 2016 IEEE
DOI 10.1109/WI.2016.49

method effectively predicts the dull nodes and links, enhancing link prediction performances.

II. Related works

A brief literature review on existing preprocessing techniques for social networks is presented next. Zhang *et. al.* [9] has proposed SocConnect, which integrates social network data collected from multiple on-line social networking sites. Network data cleaning has been considered in [10], [11], [12], [13]. Benevenuto *et. al.* [10] has proposed a method for detecting spammers in Twitter. Bhagat *et. al.* [11] has developed algorithms for anonymizing the actors in social networks in order to preserve privacy. Ferreira *et. al.* [12] has compiled a nice survey on name disambiguation methods for bibliographic citation networks. Huisman *et. al.* [13] has proposed methods to fill the missing informations in network data. Hernández *et. al.* [14] has proposed algorithms to store and retrieve large social network data in compressed format. Macskassy [15] has proposed a method for identifying nodes which leave a particular community in dynamic networks. To the best of our knowledge, none of the existing studies has attempted to predict dull nodes and links in evolving networks as a network preprocessing task.

We have found a few studies [16], [17], [18], which passively relate to the concept of dull nodes or links. Kamath *et. al.* [16] has proposed a method to track short-term group formation in on-line social networks like Twitter and Facebook. Raeder *et. al.* [17] and Miritello *et. al.* [18] have predicted "persistence" of links. Given the communication pattern of a link in a time window, their methods predict whether the link will remain active in the next window.

III. Problem formulation

This section formulates the problem of predicting dull nodes and links. Social networks typically evolve with social interactions among the actors. When two actors interact, they become connected by a link. With time, pair(s) of actors connected by a link, may interact again. We preserve this history of interaction between two actors by storing the time-stamp of occurrence of the interactions they have been involved in, and associate it to the link formed by them. We represent evolving networks up to time instant t by an undirected graph $G^t = (V, E, T, \mathcal{Q})$, where V is the set of vertices representing actor nodes; $E \subset V \times V$ is the set of edges representing links; T is the set of time-stamps of interactions occurred in the network till t; $\mathcal{Q}$ is a function $\mathcal{Q} : E \rightarrow 2^T$, which returns the historical time-stamps of all interactions associated with a link. For an example, let G^t be a Facebook wall-post network. When a user x posts on another user y's Facebook wall, they become connected by a link (x, y). All such historical wall-posts (up to time instant t) between x and y are stored and retrieved by $\mathcal{Q}(x, y)$.

Given a graph G^t, the preprocessing task is to predict the dull nodes and links, which are going to be dull at a future time t', and construct another graph G^t_r by removing these dull nodes and links from G^t.

IV. Predicting dull nodes and links

Historical activities of nodes and links are modeled to predict the dull nodes and links. First, several time-series representing historical change in baseline node and link properties are prepared and modeled using *simple exponential smoothing* [19] model. Future values of these time-series are forecast to identify features for predicting dull nodes and links. Lastly, vector distance based unsupervised method is applied to predict dull nodes and links.

A. Preparing time-series

The time-series which we are going to introduce in this subsection, are of two types: *dyadic* and *topological*. Dyadic time-series encapsulate the temporal characteristics relating *dyads, i.e.*, the historical interaction between nodes, whereas topological time-series deal with change in graph's topological measures like degree of a node, number of common neighbors of a pair of nodes etc. Given a graph G^t, time is discretized into a sequence of contiguous time-windows of constant size Δ for a particular network. A time-window ending at time $t - \Delta i$ is denoted as $w_{-i} : i \in \mathbb{N}$. All the time-stamps that fall in $\big(t-\Delta(i+1), t-\Delta i\big]$ define the window w_{-i}. w_0 represents the most recent window, and w_{-i} represents the $(i+1)^{th}$ last window. We populate the contiguous time-windows with some value $\in \mathbb{R}$ to form a time-series. The method of populating the time-windows of dyadic and topological time-series is described next.

The window $w_{-\delta}$ of a dyadic time-series $S(x, y)$, defined on a link (x, y) in G^t, is populated as:

$$S_{-\delta}(x, y) = |\mathcal{Q}(x, y) \cap w_{-\delta}|, \tag{1}$$

which gives the number of interactions between the end nodes during time-window $w_{-\delta}$. $S(x, y) = \big\langle S_{-q}(x, y), ..., S_{-1}(x, y), S_0(x, y)\big\rangle$, $q \in \mathbb{N}$, gives the dyadic time-series of the link (x, y), where w_{-q} is the time-window when the link appeared.

Similarly, the window $w_{-\delta}$ of a dyadic time-series $s(x)$, defined on a node x in G^t, is populated as the number of interactions between node x and its neighbors $\Gamma(x)$ in G^t during time-window $w_{-\delta}$:

$$s_{-\delta}(x) = \sum_{y \in \Gamma(x)} |\mathcal{Q}(x, y) \cap w_{-\delta}|, \tag{2}$$

To construct topological time-series of nodes, we consider two node properties namely *degree* and *clustering coefficient (CC)* [2]. We define time-series data for $degree$ and CC corresponding to node x in a time-window $w_{-\delta}$ as follows:

$$d_{-\delta}(x) = d^{t-\Delta\delta}(x) - d^{t-\Delta(\delta+1)}(x), \tag{3}$$

$$c_{-\delta}(x) = c^{t-\Delta\delta}(x) - c^{t-\Delta(\delta+1)}(x), \tag{4}$$

where $d^{t-\Delta\delta}(x)$ and $c^{t-\Delta\delta}(x)$ give the $degree$ and CC of node x respectively in the snapshot of G^t at $t - \Delta\delta$, $G^{t-\Delta\delta}$. We denote these two time-series as $d(x)$ and $c(x)$ respectively. Common neighbor (CN) index [7] is the baseline property

[2] CC represents the local density in the neighborhood of a node. It is quantified by the ratio of the number of interconnections among the node's neighbors, and the number of maximum possible interconnections [20].

which is used to construct the topological time-series for links. We define time-series data for CN corresponding to link (x, y) in a time-window $w_{-\delta}$ as follows:

$$C_{-\delta}(x, y) = CN^{t-\Delta\delta}(x, y) - CN^{t-\Delta(\delta+1)}(x, y), \quad (5)$$

where $CN^{t-\Delta\delta}(x, y)$ gives the common neighbor score between node-pair x and y respectively in $G^{t-\Delta\delta}$. We populate the values for $C_{-\delta}(x, y)$ in the sequence of contiguous time-windows that starts with the window where the first common neighbor of x and y appeared, and ends with w_0. This time-series is denoted as $C(x, y)$.

B. Modeling the time-series

The time-series defined in Subsection IV-A are used to forecast future trends. We apply *simple exponential smoothing* time-series forecasting method to model the time-series. The exponential smoothing method gives high importance to the recent activities, and importance decays exponentially from recent to less recent past. Let $Q_{-\delta}$ denotes the data corresponding to window $w_{-\delta}$ of a time-series Q. Forecast equation of the simple exponential smoothing model for Q can be given by following recurrence equation [19]:

$$Q'_{-\delta+1} = \alpha Q_{-\delta} + (1 - \alpha) Q'_{-\delta}, \quad (6)$$

where $Q'_{-\delta}$ gives the forecast value for time-window $w_{-\delta}$, given the time-series data present in previous time-windows; and $0 < \alpha \leq 1$ is called smoothing parameter. When $\alpha \to 0$, simple exponential smoothing gives same weightage to every window during forecast. As the value of α increases from 0 to 1, importance of the recent time-windows increases monotonically. As $\alpha \to 1$, importance of older windows decreases, and $\alpha = 1$ forecasts the value present in the last time-window. Equation 6 is used to forecast the value in the window w_1 representing the window just after time t, can be written as:

$$Q'_1 = \textstyle\sum_{i=0}^{q} \alpha(1 - \alpha)^i \; Q_{-i} + (1 - \alpha)^{q+1} Q'_{-q},$$

where $Q'_{-q} = Q_{-q}$ gives the first forecast, w_{-q} being the oldest window. α is estimated by minimizing the sum of the squared errors (SSE):

$$SSE = \textstyle\sum_{j=q-1}^{0} (Q_{-j} - Q'_{-j})^2 = \sum_{j=q-1}^{0} e_{-j}^2 = e^2,$$

where $e_{-j} = Q_{-j} - Q'_{-j}$ gives the error in window w_{-j}. Here, minimizing SSE is a nonlinear optimization task. We use vertical least square fitting procedure that estimates α by solving the equation:

$$\frac{d(e^2)}{d\alpha} = 0 \implies \textstyle\sum_{j=q-1}^{0} \dfrac{d(Q_{-j} - Q'_{-j})^2}{d\alpha} = 0.$$

C. Feature generation and unsupervised method

In this subsection, we propose an unsupervised method to predict dull nodes and links in a given network G^t, which are declared dull at a future time t'. Several features based on temporal change in dyadic and topological time-series are proposed here. These features capture the distinctive properties of a node (and a link) which is likely to be inactive or disappear, based on recent trends. As an example, from a topological perspective, if a node x does not make new connections lately, or there is a decreasing trend in making new connections, it may be a potential candidate of being dull in future. This is reflected in the time-series $d(x)$. Similarly, the trend in activity of a node x in recent times is reflected in the time-series $s(x)$. In case of links, if there is a decline in adding new common neighbors between the end nodes in recent time, they might break their relationship in near future. The forecast value of these time-series, generated using simple exponential smoothing model depicts how the nodes and links will behave in future in terms of the underlying node and link properties. We propose a number of features to predict dull nodes and links exploiting these forecast values. Table I summarizes the features proposed in this work. Suffix 1 in each of the feature values represents w_1, and the values are the forecast values for w_1 of the corresponding time-series (refer to Subsections IV-A and IV-B). $zeros_{node}$ and $zeros_{link}$ give the number of trailing zeros in dyadic time-series of the target node and the target link respectively. The $pref_{link}$ feature for a link is a combination of forecast values of the $degree$ time-series of the two end nodes. This feature is the temporal version of the preferential attachment property of real networks [21].

TABLE I. TIME SERIES DATA AND CORRESPONDING FEATURES.

	Value	Feature
Node x	$s'_1(x)$	$strength_{node}$
	—	$zeros_{node}$
	$d'_1(x)$	deg_{node}
	$c'_1(x)$	$clust_{node}$
Link (x, y)	$S'_1(x, y)$	$strength_{link}$
	—	$zeros_{link}$
	$d'_1(x)$	deg^x_{link}
	$d'_1(y)$	deg^y_{link}
	$d'_1(x) \times d'_1(y)$	$pref_{link}$
	$C'_1(x, y)$	$common_{link}$

A node or a link is described by its feature vector. *Min-Max normalization* of boundary $[0, 1]$ [3] is applied to all features. For simplicity, we use distance based unsupervised method to predict the dull nodes and links. We identify a *reference vector* that represents a dull node (and a dull link), and calculate its distance from each sample's feature vector. *Chebyshev distance* is used to measure the distance between the samples' feature vector and the reference vector. Chebyshev distance of the feature vector (say $\boldsymbol{X}$, where its elements are denoted as x_i's, i being integer values ranging from 1 to $|\boldsymbol{X}|$) and the reference vector (say $\boldsymbol{V}$, where its elements are denoted as v_i's) is calculated as follows:

$$D_{chebyshev}(\boldsymbol{X}, \boldsymbol{V}) = \max_{i=1}^{|\boldsymbol{X}|} |x_i - v_i|. \quad (7)$$

$(1 - D_{chebyshev}(\boldsymbol{X}, \boldsymbol{V}))$ gives the sample's *proximity score* with a dull node (and a dull link). The values of all features other than $clust_{node}$, $zeros_{node}$ and $zeros_{link}$ of the reference vector are set to 0, because these features should have negative correlation with the dull ones. Corresponding

[3] It reassigns a feature value v of feature V as: $\frac{v - \text{minimum value in } V}{\text{maximum value in } V - \text{minimum value in } V}$

TABLE II. DATASET SPECIFICATION

			γ	
Network	#Nodes	#Links	Node	Link
fb	44937	166511	13	18
dblp	1304128	5258985	14	10

values of $clust_{node}$, $zeros_{node}$ and $zeros_{link}$ are set to 1. After predicting the potential dull nodes and links, we remove them from G^t, for data cleaning task.

V. DATASETS

Experiments are carried out on two time-stamped social networks of different characteristics: Facebook wall-post network [22] (fb), and DBLP co-authorship network (dblp). fb consists of all the wall-posts, posted in Facebook [4] New Orleans network spanning the period September 26th, 2006 and January 22nd, 2009. The data is available with Unix time-stamp representing the time of each wall-post, and the information of who is posting on whose wall. Each wall-post is considered as an interaction. We ignore the direction of interactions in order to prepare an undirected graph so that our method can be applicable on it. dblp dataset has been downloaded from the web-link http://dblp.uni-trier.de/xml/ in .xml format. It contains the publication information in the field of computer science from the year of 1936 upto the starting of 2014. Early years contain very few publications. So, in this paper we consider only the publications which occurs during the years $1980 - 2013$. Each publication information contains at least two kinds of tags $< author >$ and $< year >$, which represent each author and the year of the publication respectively. The type of a publication is characterized by tags like $< article >$ (for a journal publication), $< inproceedings >$ (for a conference proceedings publication), $< www >$ etc. Only the conference and journal publications are used to prepare the graph. We consider each publication as an interaction, which forms a clique among all authors of that publication, and a self-loop in case of single author paper. fb and dblp consider a *month* and a *year* long time-windows respectively. A summery of characteristics of the networks is presented in Table II.

VI. DULL NODES AND LINKS

Given a network G^t, after predicting the nodes and links which will be dull at a future time $t' = t + \Delta i$, we need to evaluate our method against true dull nodes and links in $G^{t'}$. The concept of dull nodes and links is subjective, which is defined as the nodes and links which has kept silent for long, probably will never interact in future. In this section, we propose a scheme to label the dull nodes and links by using the statistical property of the network $G^{t'}$.

To label a dull node, we consider the dyadic time-series ($s(x)$) of all nodes upto time-window w_0 (with respect to $G^{t'}$). A node is labeled dull if it has not interacted with others for a long time, *i.e.*, its dyadic time-series ends with large number of consecutive zeros. Subsequently, question arises; *how large that number should be, so that we label it as a dull node?* Here we define the number by exploiting the distribution of zero-burst (a series of consecutive zeros) patterns present in all

[4]https://www.facebook.com/

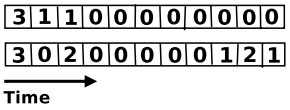


Fig. 1. A toy example.

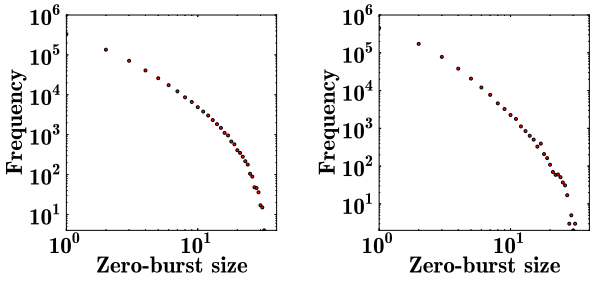
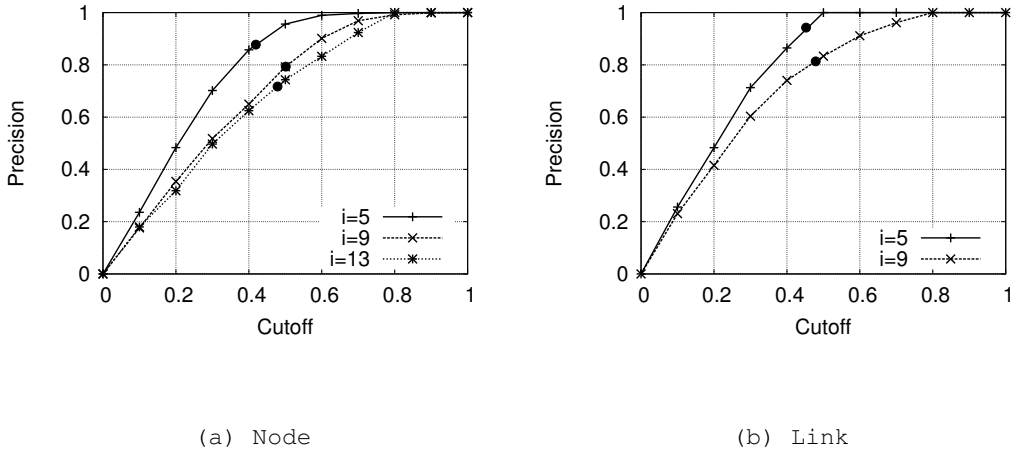


Fig. 2. Log-log plot of the frequency of zero-burst patterns for DBLP bibliographic network.

nodes' time-series. It is explained using a toy example shown in Figure 1, which presents the time-series data of two nodes in $G^{t'}$. The first one from the top has all zeros in last eight windows; so it may be labeled dull in $G^{t'}$, if a dull node is defined by at least eight or more number of consecutive zeros at the end of the dyadic time-series. The second one is having two series of intermediate zeros (or zero-burst) of lengths one and five. The tail of the distribution of such zero-burst lengths over the time-series data of all nodes in $G^{t'}$ gives a view about *watching how many consecutive zeros, one can be confident enough that the node may not interact in future, ie., is dull*. Figure 2 shows the length distribution of intermediate zero-bursts for the dblp bibliographic network. Linearity shown in the plots, which are drawn in $log - log$ scale, indicates that the distributions follow power law. This finding supports the observation of [23], where the authors have found that *inter-event time* [5] distribution follows power law. Figure 2 indicates that, in real networks, the distribution function decreases rapidly with the increase of zero-burst length, and there are few cases of high number of intermediate zero bursts. This fact encourages us to label dull node as: let $F(z)$ be the *cumulative distribution function* of the intermediate zero-burst lengths in the time-series data, over all nodes for a given network $G^{t'}$. A node x in $G^{t'}$ is labeled dull iff its time-series data is having at least γ number of consecutive zeros at the end of its dyadic time-series such that, $F(\gamma - 1) \geq \beta$, where $0 < \beta < 1$ is a constant representing the confidence level. Without loss of generality, we set $\beta = 0.99$. Similarly, the class labels for *dull link* is labeled using the dyadic time-series data of all edges in $G^{t'}$. Table II presents γ values for dblp and fb.

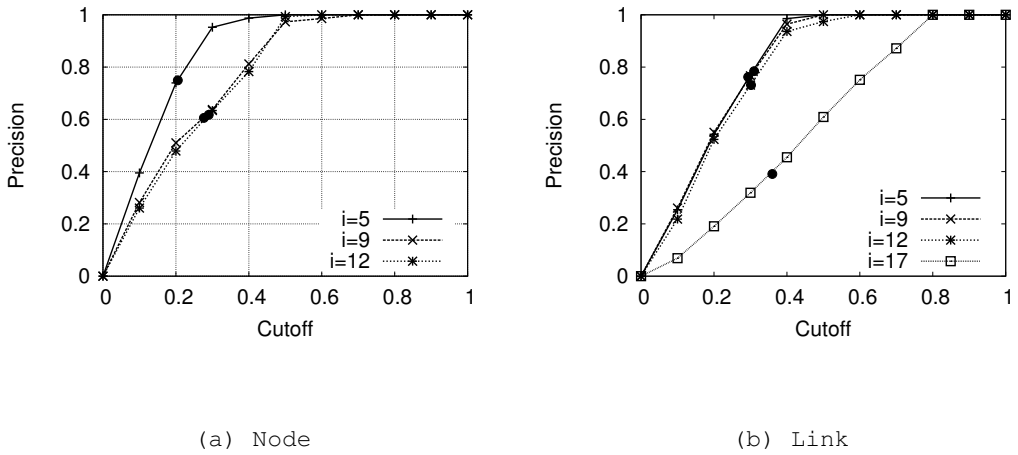
Relation between dull nodes and links: By definition, dull nodes and dull links are closely related. Let, in a given network, γ_n represents the parameter γ to define dull nodes, and γ_l represents the parameter γ to define dull links. Let x be

[5]Inter-event time is the time interval of occurrence of two consecutive events in a dynamic network.



(a) Node (b) Link

Fig. 4. Precision curve for dull nodes and links prediction in dblp.



(a) Node (b) Link

Fig. 5. Precision curve for dull nodes and links prediction in fb.

a node in the network. When $\gamma_n < \gamma_l$, if $s(x)$ has γ_n number of trailing zeros, it is possible that some of the links connecting x to its neighbors may not qualify as dull. Hence, predicting and pruning x removes those links too, which would not have been possible if predicting and pruning only the dull links was considered. Pruning the potential dull nodes adds an extra level of filtering.

VII. Evaluating proposed method

In this section, we evaluate the method proposed in Section IV. Experiments are performed over several snapshots: $G^{t'-\Delta i}, i < \gamma$, where $G^{t'-\Delta i}$ represents the given network G^t and the task is to predict the nodes and links which are declared dull at t'. Class imbalance is an inherent issue in this problem, because there are very few number of dull nodes and links as compared to the nodes and links which are not dull. We handle class imbalance by selecting only the nodes (and links) which have not interacted in window w_0 and w_{-1} (with respect to G^t) to represent the non-dull nodes (and links). In this paper, we pick top k number of samples in terms of their proximity score (described in Subsection IV-C) as potential dull nodes (and links), where k represents the number of true dull nodes (and links). Figure 3 presents the prediction performance in terms of *area under receiver operating characteristic (ROC) curve* (AUC scores) [24] for both datasets. High AUC scores in the results demonstrate that potential dull nodes and links can be predicted effectively. Prediction performance for the

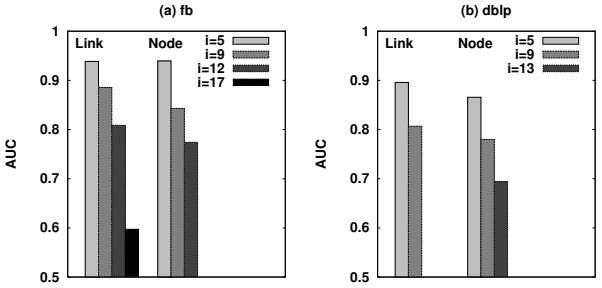


Fig. 3. Performance of dull nodes and links prediction at time $t' - \Delta i$ in terms of AUC score.

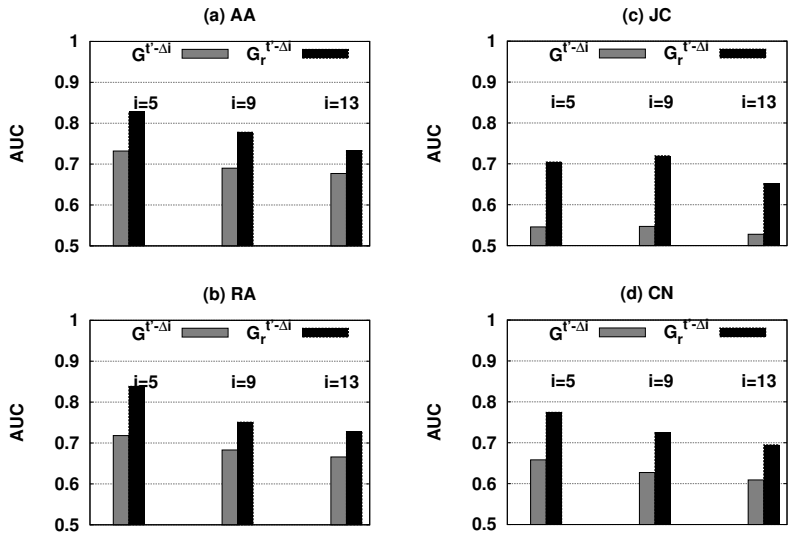


Fig. 6. Link prediction performance (dblp)

lower values of i is much better than the higher ones, because a high number of zeros at the end of all time-series of the dull samples boosts the performance when the value of i is small. Figures 4 and 5 present the prediction performance in different thresholds in terms of their *precision* scores. The black dots in each plot represent $precision@k$ values, k representing the number of true dull ones (the pruning threshold). The precision plots demonstrate that the proposed method achieves very high precision before reaching 0.5 cutoff value. As we set k as the cutoff in our prediction mechanism, $precision@k$ will govern the performance of our method in practical scenario, which is basically a retrieval process. The plots show that our method can achieve high $precision@k$ values. Although rise of the precision curves for fb is more rapid than dblp, $precision@k$ values win in dblp. It is also notable that, in spite of overcoming class imbalance, the proportion of dull nodes and links are much less in fb, which is another cause of performance degradation at the retrieval point.

VIII. Data cleaning and link prediction: a case study

After predicting the dull nodes and links from $G^{t'-\Delta i}$, we investigate the effect of cleaning the datasets by removing those nodes and links, on identifying new links that will appear in $G^{t'}$. We first sort the proximity scores for all samples (links and nodes) and consider a number of best performing k samples, where k represents the number of labeled dull nodes and links, as the potential candidate of being dull. We prepare a graph $G_r^{t'-\Delta i}$ by removing the predicted dull nodes and links from $G^{t'-\Delta i}$. We perform link prediction on both of $G^{t'-\Delta i}$ and $G_r^{t'-\Delta i}$ using state-of-the-art link prediction methods, and compare their performance in terms of AUC scores. We consider four baseline link prediction methods: *Common neighbor (CN), Jaccard's coefficient (JC), Adamic/Adar (AA), and Resource allocation (RA)* [7], [25]. These methods are described in Table III. We perform the experiments for multiple values of i. Figures 6 and 7 summarizes the link prediction results for dblp and fb. These figures show that prediction performance in the reduced graph $G_r^{t'-\Delta i}$ improves significantly for almost all values of i's over both the datasets. dblp network is benefited more than fb by data cleaning. It can be justified by its higher $precision@k$ values in dull link prediction (refer to Section VII). In most of the cases, over all link prediction methods and datasets, effect of data cleaning increases with increase in the value of i. In fb, sometimes data cleaning worsens the link prediction when $i = 17$. It is expected because, as shown in Figure 5, $precision@k$ value for dull link prediction is as low as .39 in $G^{t'-17\Delta}$. Among the link prediction methods, JC and CN are the most affected prediction methods, whereas AA and RA are more robust against noise.

TABLE III. State-of-the-art link prediction methods.

Method	Definition
CN	$\lvert\Gamma(x) \cap \Gamma(y)\rvert$
JC	$\frac{\lvert\Gamma(x)\cap\Gamma(y)\rvert}{\lvert\Gamma(x)\cup\Gamma(y)\rvert}$
AA	$\sum_{z\in\Gamma(x)\cap\Gamma(y)} \frac{1}{log\lvert\Gamma(z)\rvert}$
RA	$\sum_{z\in\Gamma(x)\cap\Gamma(y)} \frac{1}{\lvert\Gamma(z)\rvert}$

Note: $\Gamma(a)$ denotes the set of neighbors of node a.

IX. Conclusion

In this paper, we have proposed a time aware network data cleaning method. The proposed method predicts dull nodes and links, based on historical network properties. It then prunes them from the network. This paper has also proposed a scheme to label dull nodes and links for evaluating the proposed method. Exhaustive experiments on two real network datasets have shown that the proposed method can effectively predict the potential dull nodes and links. It further

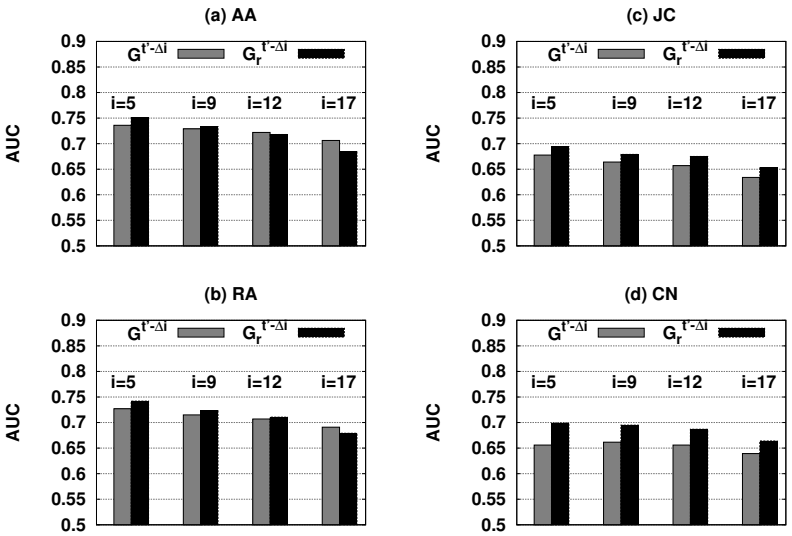


Fig. 7. Link prediction performance (fb)

has demonstrated the effect of network data cleaning on state-of-the-art link prediction methods. Significant improvement in link prediction performance has been observed when the proposed data cleaning method is applied on real datasets.

REFERENCES

[1] A.-L. Barabâsi, H. Jeong, Z. Néda, E. Ravasz, A. Schubert, and T. Vicsek, "Evolution of the social network of scientific collaborations," *Physica A: Statistical mechanics and its applications*, vol. 311, no. 3, pp. 590–614, 2002.

[2] G. Kossinets and D. J. Watts, "Empirical analysis of an evolving social network," *Science*, vol. 311, no. 5757, pp. 88–90, 2006.

[3] T. Tylenda, R. Angelova, and S. Bedathur, "Towards time-aware link prediction in evolving social networks," in *SNA-KDD '09*. New York, NY, USA: ACM, 2009, pp. 9:1–9:10.

[4] E. Richard, S. Gaïffas, and N. Vayatis, "Link prediction in graphs with autoregressive features," *J. Mach. Learn. Res.*, vol. 15, no. 1, pp. 565–593, Jan. 2014.

[5] D. Chakrabarti, R. Kumar, and A. Tomkins, "Evolutionary clustering," in *Proceedings of the 12th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2006, pp. 554–560.

[6] Y. Chi, X. Song, D. Zhou, K. Hino, and B. L. Tseng, "On evolutionary spectral clustering," *ACM Trans. Knowl. Discov. Data*, vol. 3, no. 4, pp. 17:1–17:30, dec 2009.

[7] D. Liben-Nowell and J. Kleinberg, "The link-prediction problem for social networks," in *Conference on Information and Knowledge Management (CIKM03)*, 2003, pp. 556–559.

[8] M. Girvan and M. E. J. Newman, "Community structure in social and biological networks," *Proceedings of the National Academy of Sciences*, vol. 99, no. 12, pp. 7821–7826, 2002.

[9] J. Zhang, Y. Wang, and J. Vassileva, "Socconnect: A personalized social network aggregator and recommender," *Information Processing & Management*, vol. 49, no. 3, pp. 721–737, 2013.

[10] F. Benevenuto, G. Magno, T. Rodrigues, and V. Almeida, "Detecting spammers on twitter," in *Collaboration, electronic messaging, anti-abuse and spam conference (CEAS)*, vol. 6, 2010, p. 12.

[11] S. Bhagat, G. Cormode, B. Krishnamurthy, and D. Srivastava, "Prediction promotes privacy in dynamic social networks," in *Proceedings of the 3rd conference on Online social networks*, 2010, pp. 6–6.

[12] A. A. Ferreira, M. A. Gonçalves, and A. H. Laender, "A brief survey of automatic methods for author name disambiguation," *Acm Sigmod Record*, vol. 41, no. 2, pp. 15–26, 2012.

[13] M. Huisman and C. Steglich, "Treatment of non-response in longitudinal network studies," *Social networks*, vol. 30, no. 4, pp. 297–308, 2008.

[14] C. Hernández and G. Navarro, "Compressed representations for web and social graphs," *Knowledge and information systems*, vol. 40, no. 2, pp. 279–313, 2014.

[15] S. A. Macskassy, "Mining dynamic networks: The importance of pre-processing on downstream analytics," *COMMPER 2012*, p. 2, 2011.

[16] K. Y. Kamath and J. Caverlee, "Transient crowd discovery on the real-time social web," in *Proceedings of the fourth ACM international conference on Web search and data mining*. ACM, 2011, pp. 585–594.

[17] T. Raeder, O. Lizardo, D. Hachen, and N. V. Chawla, "Predictors of short-term decay of cell phone contacts in a large scale communication network," *Social Networks*, vol. 33, no. 4, pp. 245–257, 2011.

[18] G. Miritello, *Temporal patterns of communication in social networks*. Springer Science & Business Media, 2013.

[19] R. Hyndman, A. Koehler, J. Ord, and R. Snyder, *Forecasting with Exponential Smoothing: The State Space Approach*, ser. Springer Series in Statistics. Springer, 2008.

[20] D. J. Watts and S. H. Strogatz, "Collective dynamics of small-world networks," *Nature*, vol. 393, no. 6684, pp. 440–442, 1998.

[21] M. E. Newman, "Clustering and preferential attachment in growing networks," *Physical review E*, vol. 64, no. 2, p. 025102, 2001.

[22] B. Viswanath, A. Mislove, M. Cha, and K. P. Gummadi, "On the evolution of user interaction in facebook," in *WOSN '09*. New York, NY, USA: ACM, 2009, pp. 37–42.

[23] M. Karsai, K. Kaski, A.-L. Barabási, and J. Kertész, "Universal features of correlated bursty behaviour," *Scientific reports*, vol. 2, 2012.

[24] J. A. Hanley and B. J. Mcneil, "The meaning and use of the area under a receiver operating characteristic (ROC) curve." *Radiology*, vol. 143, no. 1, pp. 29–36, april 1982. [Online]. Available: http://radiology.rsnajnls.org/content/143/1/29.abstract

[25] T. Zhou, L. Lü, and Y.-C. Zhang, "Predicting missing links via local information," *The European Physical Journal B - Condensed Matter and Complex Systems*, vol. 71, no. 4, pp. 623–630, 2009.

FLIPPER: Fault-Tolerant Distributed Network Management and Control

Subhrendu Chattopadhyay
Department of CSE
IIT Guwahati, India 781039
subhrendu@iitg.ernet.in

Niladri Sett
Department of CSE
IIT Guwahati, India 781039
niladri@iitg.ernet.in

Sukumar Nandi
Department of CSE
IIT Guwahati, India 781039
sukumar@iitg.ernet.in

Sandip Chakraborty
Department of CSE
IIT Kharagpur, India 721302
sandipc@cse.iitkgp.ac.in

***Abstract*—The current developments of software defined networking (SDN) paradigm provide a flexible architecture for network control and management, in the cost of deploying new hardwares by replacing the existing routing infrastructure. Further, the centralized controller architecture of SDN makes the network prone to single point failure and creates performance bottleneck. To avoid these issues and to support network manageability over the existing network infrastructure, we develop Flipper in this paper, that uses only software augmentation to convert existing off-the-shelf routers to network policy design and enforcement points (PDEP). We develop a distributed self-stabilized architecture for dynamic role change of network devices from routers to PDEPs, and make the architecture fault-tolerant. The performance of Flipper has been analyzed from both simulation over synthetic networks, and emulation over real network protocol stacks, and we observe that Flipper is scalable, flexible and fail-safe that can significantly boost up the manageability of existing network infrastructure.**

***Index Terms*—Network architecture, Fault-tolerance, Programmable network, software defined network**

I. Introduction

Consider the following scenario. The network administrator of an academic institute wants to dynamically update the bandwidth distribution policies based on network usage statistics. The institute network is connected with multiple network service providers, and therefore she needs to update the configuration at different edge routers and gateways. With traditional network devices, like layer 3 switches, this task is tedious, as even a minor configuration inconsistency among the edge routers and gateways may lead to severe network underutilization or bandwidth imbalance. Further, the system is also not scalable for such dynamic updates of network configuration policies.

Software defined networking (SDN) [1] is a technique, which can help in dynamic network configuration update. SDN uses centralized controller to convert policies to device configurations, and to update the targeted devices in the network with the corresponding configurations. To enforce policy to configuration conversion, SDN segregates the network control plane from the data plane. The SDN control plane takes care of all the control functionalities and update of device parameters and configurations, whereas the data plane is only responsible for data forwarding based on the configuration parameters set by the control plane.

Although SDN has revolutionized dynamic network management aspects, it requires specific hardwares that can understand the language for SDN, like OpenSwitch [2], [3], so that a SDN controller can dynamically update the configuration parameters for those hardwares. Therefore the important question is: *How much effort and cost do one need to convert an existing network infrastructure to a SDN supported one?* The existing studies in this direction talk about interoperability among SDN supported and non-SDN network devices, such that incremental deployment of SDN supported devices becomes possible [4]–[6]. However, the concern about cost-effectiveness is still there. SDN supported hardwares are much costlier than commercial off-the-shelf (COTS) network devices, and therefore requires huge operational expenditure to replace existing infrastructure by SDN supported infrastructure.

Although it is quite inevitable that the future of network management is SDN, but simultaneously we also ask this question: *Can we make our existing network more management friendly, such that dynamic network configuration becomes possible without much changing the existing infrastructure?* This paper tries to find out the answer of this question. We show that it is quite possible to use the existing COTS routers to work as network policy decision and enforcement points (PDEP), which are known as network information base (NIB). We can turn a COTS router to a NIB by installing a few additional software tools to support "network function virtualization" [7] (NFV). With the help of NFV functionalities, a COTS router can dynamically update the policy control parameters within its neighborhood [8], [9]. Accordingly, we develop a new network management architecture, which is somewhere in-between the traditional architecture and SDN based architecture, where the COTS routers dynamically change their roles from a conventional network router to a NIB, and participate in PDEP functionalities. We call this architecture *Flipper*.

Flipper has two specific advantages over SDN based network architecture, among others. First, to implement Flipper, a network administrator does not need to procure new costly hardwares, and second, Flipper avoids the controller bottleneck problem [4], [7], [10]–[12] which is much debated in the SDN research community. Flipper is a distributed architecture, where the COTS routers execute a distributed self-stabilizing algorithm to decide which nodes can work as a NIB. As

978-3-901882-89-0 @2017 IFIP

the NIBs have limited resources because they are built on top of the existing routers, a NIB can manage, control and update the network policies only among its neighborhood. Therefore, we develop a distributed self-stabilizing maximal independent set (MIS) selection mechanism, which is indeed non-trivial. To maintain consistency in policy decisions across the network, we have developed a fault-tolerant NIB selection mechanism. We analyse the closure, fault-tolerance and scalability properties of Flipper. The performance of Flipper is analysed from both simulation through a synthetic network environment, as well as through real implementation over an emulation platform using *network name-space*. Our implementation of Flipper provides a proof-of-concept support of the new architecture, while compare the performance with that of other protocols in terms of flow initiation delay.

II. Flipper Architecture

This section gives the details of Flipper architecture and its working procedure. Flipper uses a technology to convert existing COTS routers to PDEP devices through NFV. For this task to convert a COTS router to a PDEP supported device, we use the existing technology called ONIX [7] that describes how the NFV modules can be interfaced with existing router operating system to make it work as a PDEP device that can sync up network policies with other PDEP devices, converts it to network configurations and feeds up those configurations to other normal routers in the neighborhood. Although we use the existing ONIX technology for this purpose, but deploying it over an existing network is non-trivial, because of the limited processing capacity of the existing COTS routers. As a consequence, such devices introduce large delay and processing overhead if a single ONIX node works like a SDN controller. Therefore, in Flipper, we introduce a distributed dynamic PDEP selection mechanism, which is self-stabilized and fault-tolerant. The details of this architecture is discussed next.

A. What is Flipper?

Our proposed Flipper architecture consists of following components which are similar to ONIX. Although the components are similar, the functionalities and arrangement of the components are completely different in Flipper.

1. OpenFlow supported switch (OFS): An OFS is responsible for data forwarding based on forwarding rule set. OpenFlow [13], [14] is a software component that is installed in the router OS to provide NFV functionalities. However, mere OpenFlow support does not make these devices SDN complaint, as specialized hardware (like OpenSwitch) is required for this purpose. In our Flipper architecture, we install additional components only at the software level, but the COTS hardwares are used.

2. Host: End user devices connected with OFSs that hosts the applications and generates data traffic.

3. DHT-NIB: Memory based high update prone eventually-consistent *"distributed hash table"* based NIB (DHT-NIB) for storing link level information of switches. DHT-NIB also helps

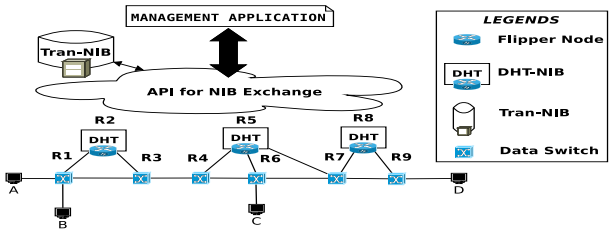


Fig. 1: FLIPPER: Architecture

in setting up forwarding rules in switches based on control application. As shown in ONIX architecture, an OFS can act as a DHT-NIB with additional functionalities.

4.tran-NIB: Strongly consistent tran-NIB is used for rarely changed network wide policy management.

A major difference between the existing SDN based architecture and Flipper is that the standard SDN components have fixed roles to play. However, in this paper we define a Flipper device as a service grade router which can dynamically choose a role of either OFS or DHT-NIB. This dynamic change of roles (*"flip"*) are possible due to use of NFV in Flipper devices. For the sake of readability we refer the Flipper architecture as *"FLIPPER"* and Flipper devices as *"flipper"*.

B. How FLIPPER works?

To understand the working principle of FLIPPER, we take help of Figure 1. The topology consists of a dedicated high performance transactional NIB, hosts (A, B, C, D) and flippers $(R1, R2, ..., R9)$. *"switch-flipper"* if a flipper that acts as a OFS. *"DHT-flipper"* are the flippers that perform DHT-NIB functionalities. Initially, flippers adjust themselves so that a switch-flippers have at least one DHT-flipper in its neighborhood. Upon receiving a flow request from switch-flipper, the distributed control plane consults the relevant DHT-flippers based on the programmable network rules in tran-NIBs and completes the flow table setup procedure in switch-flippers.

C. Fault-tolerance in FLIPPER

The use of NFV for deployment of services provides the flexibility towards FLIPPER. However, general purpose switches in a service provider network are failure prone. The failure of a DHT-flipper can significantly affect the network performance as it controls all the flows in its neighborhood. Therefore, to maintain the robustness of the architecture, FLIPPER needs to be fault-tolerant. For example, in Figure 1, let us assume that $R3, R5$ and $R8$ are acting as DHT-flippers. The associated switch-flippers of $R2, R5$ and $R8$ are $\{R1, R3\}, \{R4, R6, R7\}$ and $\{R9\}$, respectively. If $R5$ fails, $R4, R6$ can not work in the absence of DHT-flipper. For maintaining fault tolerance, we propose a distributed flipper readjustment framework. Whenever one or more switch-flippers detect unavailability of DHT-flipper in its (their) neighborhood, it (they) invokes (invoke) flipper readjustment procedure. The re-adjustment procedure provides the newly selected set of DHT-flippers and switch-flippers. After reaching a consensus, each switch-flipper notifies its adjacent DHT-flipper with its

state information. A switch-flipper having multiple DHT-flippers in its neighborhood chooses a DHT-flipper randomly. Therefore, they can initiate the distributed flipper readjustment framework. In this case, we use distributed self stabilization technique to make the flipper readjustment fault-tolerant.

III. Fault-tolerant Flipper Readjustment

To make the readjustment of switch-flippers and DHT-flippers fault-tolerant, we consider the use of *"self-stabilization"* [15] which is a popular technique to provide defense against *"transient failures"*. A transient failure is defined as irregular and unpredictable brief failure. In this work, we propose a novel flipper readjustment algorithm which expectedly converges with linear time complexity. Our proposed algorithm also satisfies the basic properties of self-stabilization which are as following.

1) **Convergence:** From any state, the system must reach a legitimate or desired state eventually.
2) **Closure:** In case of no failure, the system is guaranteed to remain in legitimate states.

We consider the network as a graph $G = \{V, E\}$, where V is the set of flippers, and E is the set of the edges representing physical connections among flippers. Each flipper periodically senses the physical medium for detecting link failure. A flipper i maintains label $Label_i = \{NIB, Swi, Wait\}$ and priority variable $Pri_i = \{0, 1, \ldots, B\}$, where B denotes the maximum degree of G. Any flipper k with $Label_k = NIB$ signify that, the flipper k is ready to act as a DHT-flipper. Similarly, a flipper l with $Label_l = Swi$ acts as an switch-flipper. We consider flipper with $Label = Wait$ as a flipper with intermediate state whose role is yet to decide. Neighborhood of flipper i is denoted by N_i. Flipper i also maintains $N_i^{NIB} = \underset{j \in N_i}{\forall} j : Label_j = NIB$ and $N_i^{Wait} = \underset{j \in N_i}{\forall} j : Label_j = Wait$. We consider the state of i as $(Label_i, Pri_i)$. Each flipper also maintains the state of its adjacent neighbor flippers. When a flipper changes its state, it pro-actively notifies its neighbors. Upon detecting a link failure, flipper removes the entry about the corresponding neighbor from its table.

We represent our proposed algorithm as a set of guarded actions, where each guarded action is termed as a *rule*. A rule R_j, uses the following representation, $(\mathbf{R_j})|< \mathbf{G_j} > \longrightarrow < \mathbf{A_j} >$, where $< \mathbf{G_j} >$ represents the condition which is required to be satisfied to execute action $< \mathbf{A_j} >$. Upon receiving an update from the neighbor, each flipper checks the guard statements of the rules. If any one of the guard is found to be true, then the corresponding action is executed.

A. flipper Readjustment in Case of Failure

Following the aforementioned model, the flipper readjustment problem is defined as follows. Given a network graph G, the objective of the flipper readjustment problem is to find the set of DHT-flippers in such a way that all switch-flippers can have at least one DHT-flipper in their neighborhoods, so that the policy updates can be done with minimal control plane delay and network overhead. The solution approach must find

Variables:

$\mathbf{Label_i} = \{\mathbf{NIB}, \mathbf{Swi}, \mathbf{Wait}\}$ $\quad$ $\mathbf{Pri_i} = \{\mathbf{0}, \mathbf{1}, \ldots, \mathbf{B}\}$

Functions:

$\mathbf{N^{NIB}(i)} = \forall \mathbf{j} \in \mathbf{N_i} : \mathbf{Label_j} = \mathbf{NIB}$ $\quad$ $\mathbf{N^{Wait}(i)} = \forall \mathbf{j} \in \mathbf{N_i} : \mathbf{Label_j} = \mathbf{Wait}$

$\mathbf{Max_W(i)} = \forall \mathbf{j} \in \mathbf{N_i^{Wait}} : \mathbf{Max(Pri_j)}$ $\quad$ $\mathbf{Trial(i)} : \mathbf{Pri_i} = \mathbf{Rand(0, 1, 2, \ldots, B)}$

Rules:

$(\mathbf{R_1})$—$(\mathbf{Label_i} = \mathbf{Swi})$ $\bigwedge(\mathbf{N^{NIB}(i)} = \emptyset)$ $\longrightarrow$ $(\mathbf{Label_i} = \mathbf{Wait})|\mathbf{Trial}$

$(\mathbf{R_2})$—$(\mathbf{Label_i} = \mathbf{NIB})$ $\bigwedge(\mathbf{N^{NIB}(i)} \neq \emptyset)$ $\longrightarrow$ $(\mathbf{Label_i} = \mathbf{Swi})$

$(\mathbf{R_3})$—$(\mathbf{Label_i} = \mathbf{Wait})$ $\bigwedge(\mathbf{N^{NIB}(i)} \neq \emptyset)$ $\longrightarrow$ $(\mathbf{Label_i} = \mathbf{Swi})$

$(\mathbf{R_{4a}})$—$(\mathbf{Label_i} = \mathbf{Wait})$ $\bigwedge(\mathbf{N^{NIB}(i)} = \emptyset)$ $\bigwedge(\mathbf{Pri_i} = \mathbf{Max_W(i)})$ $\longrightarrow$ $(\mathbf{Label_i} = \mathbf{Wait})|\mathbf{Trial}$

$(\mathbf{R_{4b}})$—$(\mathbf{Label_i} = \mathbf{Wait})$ $\bigwedge(\mathbf{N^{NIB}(i)} = \emptyset)$ $\bigwedge(\mathbf{Pri_i} > \mathbf{Max_W(i)})$ $\longrightarrow$ $(\mathbf{Label_i} = \mathbf{NIB})$

Here $\emptyset$ represents empty set.

Fig. 2: SS-MIS Protocol

an alternative DHT-flipper dynamically when any flipper or link fails, to incorporate fault-tolerance property. The flipper readjustment mechanism is similar to finding a *"maximal independent set"* (MIS) in flipper connectivity graph. We propose a novel distributed anonymous "self-stabilizing MIS" (SS-MIS) algorithm to find DHT-flippers dynamically. The reason for using anonymous algorithm is to remove the unfairness issue caused by the identifier system. Our proposed anonymous SS-MIS protocol has a step complexity [1] of $\mathcal{O}(n)$. Although there exists a linear time self-stabilizing distributed algorithm [16] for solving MIS problem, to the best of our knowledge in case of anonymous systems the best proposed solution [17] has $\mathcal{O}(n \log n)$ step complexity. In this work, we propose a linear time algorithm for anonymous systems that can significantly reduce the control plane overhead in FLIPPER.

B. SS-MIS Algorithm for flipper Readjustment

The proposed SS-MIS protocol selects switch-flippers and DHT-flippers in terms of assigning $Label = Swi$ and $Label = NIB$ respectively. According to MIS properties, no two DHT-flipper can be adjacent and, each switch-flipper should have at least one DHT-flipper in its adjacency list. The proposed protocol is described in Figure 2.

A flipper i which has $Label_i = Wait$ or $Label_i = NIB$, violates the independence property if any of its neighbor is in NIB state. Hence, it must execute (R_2, R_3), and must go to a state having $Label_i = Swi$. If two adjacent flipper have $Label = Wait$, and no other neighbor of them are in $Label = NIB$ state, then both the adjacent flippers will try to enter in a state with $Label = NIB$ state which requires a tie breaking mechanism. Although, the tie breaking can be done using an identifier (ID) of the flipper, in this work ID based tie breaking is not used. ID based tie breaking introduces unfairness problem, because a flipper with higher ID always gets a priority. Therefore, to break this tie, a randomized trial is performed. The proposed random trial is designed in the following way. Each node in $Wait$ state generates a random number in the range $\{0, 1, 2, \ldots, B\}$, and assigns to Pri. A

[1]Execution of an action is called a step. Step complexity of a distributed system is defined as the number of steps executed by the system. Throughout this work, the terms step and move are used invariably.

"Winner" is decided based on the unique maximum Pri value in a closed neighborhood. If no winner is found in a single experiment, it is repeated until there is a winner. The winner gets the privilege to enter into the NIB state.

IV. Properties of Flipper Architecture

In this section we discuss about the properties of proposed flipper architecture. Let the global state of the system be denoted as $\mathcal{S}$; and *legitimate state* is defined as the global configuration where no further rule may be applied at any flipper. We claim that the proposed scheme is self-stabilizing. A proof of self-stabilization requires the proof of **Closure property** and **Convergence property**.

A. FLIPPER Supports Closure Property

Theorem 1: If any flipper in the system is in intermediate state then there is at least one rule which can be executed.

Proof: Assume the state of an intermediate flipper u is $Wait$. Now there can be following scenarios.

Case 1: $\exists v \in N_u : (Label_v = NIB)$. In this case, R_3 will be applicable.

Case 2: $\forall v \in N_u : (Label_v = Wait)$ and $(Pri_v < Pri_u)$. In this case flipper u has unique maximum priority. R_{4b} will be applicable on flipper u and it will act as DHT-flipper.

Case 3: $\exists v \in N_u : (Label_v = Wait)$ and $(Pri_v = Pri_u)$ where Pri_v and Pri_u are maximum in their neighborhood. In this case flipper u and v must apply rule R_{4a} and retrial for a new priority value.

Case 4: $\exists v \in N_u : (Label_v = Wait)$ and $(Pri_v > Pri_u)$. Also $\exists w \in \mathcal{N}(v) : (Label_w = Wait)$ and $(Pri_w > Pri_v)$. From this statement it can be concluded that $(Pri_w > Pri_u)$. Hence priority of these forms a non-increasing function. Also number of flippers are bounded by N. Hence, at least one flipper will have highest priority which will be able to execute rule R_{4b} or R_{4a}. ∎

Corollary 1.1: **(Closure property)** If the system is in a state where flippers with DHT-flippers form a MIS, it will remain in that state forever, provided no further fault occurs.

Corollary 1.1 also suggests the correctness of the proposed scheme. Detailed proof of this statement is omitted due to space restriction of the paper.

B. FLIPPER Converges If a Failure Occurs and It is Scalable

A self-stabilizing system always converges in case of a failure. We analyze the algorithm and prove that, the expected time required to converge is linearly dependent on the number of flipper used.

Theorem 2: Let N denote the cardinality of the closed neighborhood of any arbitrary flipper v. Let $P(N, B)$ denote the probability of finding an unique maximum in the closed neighborhood of v. Then

$$P(N,B) = \frac{(N \times \sum_{i=1}^{B} i^{(N-1)})}{(B+1)^N}$$

Proof: Let i be the highest priority in a configuration $\mathcal{S}$ after one round, where each round corresponds to the event of generating the priority by at most each flipper in the closed neighborhood of v. To satisfy unique maximum property, i can be assigned to any one of the N flippers and the rest of the flippers can have a priority value ranging from 0 to $i-1$. So there will be $N \times i^{(N-1)}$ different possibilities. The value of i can vary from 1 to B. The sample space is $(B+1)^N$ as each node in the closed neighborhood of v can take values from 0 to B independently. Hence the total probability:

$$P(N,B) = \frac{(N \times \sum_{i=1}^{B} i^{(N-1)})}{(B+1)^N}$$

∎

Consider N flippers in the closed neighborhood of v are executing R_{4a} and R_{4b}. To find the expected number of rounds for one of the intermediate flipper to move to DHT-flipper state, we have to find the expected number of rounds in which there will be one flipper with unique maximum Pri in the neighborhood.

Theorem 3: If X denote the random variable indicating the number of rounds required to find a unique maximum priority in the closed neighborhood of v then $E[X] \leq e$, where e represents *"Euler-Mascheroni constant"*.

Proof: For calculating expected number of rounds, we need to determine the probability distribution function

$$Pr[X = r] = (1 - P(N,B))^{(r-1)} \times P(N,B)$$

Clearly this is a geometric distribution and we can say that the expected number of rounds can be calculated as following

$$E[X] = \frac{1}{P(N,B)} = \frac{(B+1)^N}{N \times \sum_{i=1}^{B} i^{N-1}}$$

$$E[X] \leq \frac{(B+1)^{B+1}}{(B+1) \times \int_0^B i^B \, di} = \frac{(B+1)^{B+1}}{B^{B+1}} = \left(1 + \frac{1}{B}\right)^{(B+1)}$$

Note here that the value of N is upper bounded by $(B+1)$. Hence

$$\lim_{B \to \infty} \left(1 + B^{-1}\right)^{(B+1)} = e$$

Therefore $E[X] \leq e$. This result signifies that each flipper needs *"e"* moves on average for the transition from $Label = Wait$ state to $Label = NIB$ state. ∎

Theorem 4: Only the following sequence or subsequence of state change is possible for each flipper during the execution of the protocol.
(Wait→Swi→Wait→Swi), (Wait→Swi→Wait→NIB), (NIB→Swi→Wait→Swi), (NIB→Swi→Wait→NIB)

Proof: We can see in Figure 2 that if a flipper executes R_{4b} then it will not execute any other rule. So no other flipper in its neighborhood can go to $Label = NIB$ state. It can also be shown that if a flipper executes R_{4b} then its neighbors can only execute R_2.

Now from Theorem 3 we can say that each node takes expected e moves to go from $Label = Wait$ state to $Label = NIB$ state. Hence the sequences will take expected $2 + e$ moves. This is true for each flipper. Therefore, we can conclude $\mathcal{O}(n)$ is the expected number of moves for convergence. ∎

Being a self-stabilized algorithm flippers are most of the time available except the convergence time. We have also shown

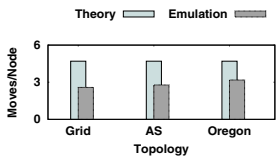


Fig. 3: Moves per flipper

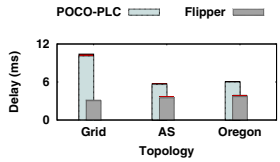


Fig. 4: Flow setup delay

that, the expected convergence time is also within a finite and acceptable bound. Therefore, we can argue that FLIPPER is scalable.

C. FLIPPER is Partition Tolerant

A partition tolerant network can function individually and independently, even if it gets partitioned due to link or node failure. As flipper readjustment does not require any bootstrapping, therefore the proposed architecture is partition tolerant. FLIPPER requires each switch-flipper to have atleast one DHT-flipper in their neighborhood. Corollary 1.1 ensures this property. Therefore, even the network becomes partitioned due to failure, FLIPPER helps them to function individually.

V. Analysis of Flipper Performance from Simulation over Synthetic Networks

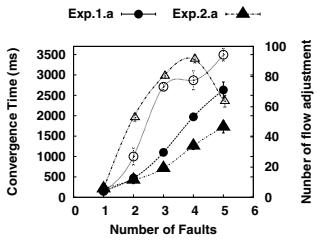
To evaluate the performance of FLIPPER, we simulate the proposed method and compared with one standard fault resilient SDN based framework, called POCO-PLC [18]. POCO-PLC is a distributed SDN platform that uses 20% of controller nodes to provide a Pareto optimal fault resiliency. The controllers act as NIB also. However, POCO-PLC provides an off-line solution of controller placement problem. On the other hand, POCO-PLC can handle limited node failure, whereas FLIPPER can sustain arbitrary node failures.

A. Simulation Setup

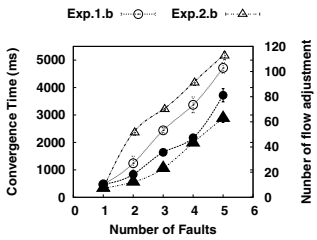
For simulation we use NS-3.22 [19] network simulator. We use three different topologies. Topology 1 is a synthetic 64×64 regular grid topology. Topology 2 (AS Topology [20]) and Topology 3 (Oregon [21]) are real autonomous system data sets taken from University of Oregon Route Views Project BGP logs, where each node represents a border router. For simulation purpose, we consider that these border routers are flipper devices. In each case, flippers are connected via 100 Mbps capacity and 2 ms delay Ethernet channels. Each flipper is configured to generate 4 flows/second with 5 Mbps data rate.

B. Results and Analysis

Figure 3 depicts the average number of moves executed by a flipper in case of random number of flipper failures when SS-MIS is used. It shows that the simulation results do not exceed the theoretical expected bound, which is $2 + e$ (see Theorem 4). The number of used DHT-flipper depends not only on the number of nodes but also on the topology itself. We found that, the required number of DHT-flippers for the two real dataset does not exceed 30%. This result is optimistic in a sense that, if the existing network infrastructure is to be deployed, then $25\% - 30\%$ of the total number of flippers are required to act as DHT-flipper for reducing flow set-up



(a) Effect of Flipper Failure



(b) Link Failure

Fig. 5: Effect of Failure

TABLE I: Emulation Topology Properties

Attributes	Values	Attributes	Values
Nodes	50	Edges	136
Avg Degree	5.44	Max Degree	20
Average DHT-flippers	18 ± 1.23	Average flow set-up delay	42.29 ± 7.2ms

delay. Figure 4 presents a comparison between the proposed flipper architecture and the existing POCO-PLC framework in terms of flow setup delay. The POCO-PLC framework uses a heuristic Pareto-optimal solution for distributed controller placement. Their work suggests the delay will be Pareto-optimal in case of 20% controller usage in most of the network scenarios. However, Figure 4 shows that, in case of flipper, $5\% - 10\%$ increase in number of controllers can reduce flow setup delay by more than 60% for both of the real networks. The performance improvement in terms of flow setup delay is due to the fact that, each switch-flipper has a DHT-flipper in it's neighborhood.

VI. Analysis of FLIPPER from Emulation over a Testbed

Motivated by the simulation results, we have emulated our proposed architecture on top of *mininet* [22]. *mininet* creates virtual nodes for emulated environments over a real networking testbed.

A. Testbed Setup

We have taken a 50 node topology extracted from Oregon dataset [21]. Each node is configured to act a switch-flipper and DHT-flipper with the help of existing OpenVSwitch [23] and OpenVSwitch database server [24] respectively. The flippers are connected via links of 5 Mbps and 2 ms delay. The link characteristics are configured with Linux *"tc"* utility. Each node generates 4 random TCP flows consuming 5 Mbps of bandwidth each. The rest of the topology characteristics and the cumulative results from the emulation are given in Table I. Each flipper periodically checks for the states of adjacent neighbors and links within a time period of 20ms. When a DHT-flipper fails, the newly appointed DHT-flipper interacts with switch-flippers via *"JSON-rpc"* and gathers flow table as well as link state information. The objective of these experiments is to identify the effect of different types of failures on data plane operation. As POCO-PLC uses static role assignment, the convergence time of the protocol becomes irrelevant. Therefore, we do not compare flipper with POCO-PLC in emulated experiments.

B. Effect of Node Failure

We select variable number of flippers as candidates for failure. To visualize the effect of mutual separation (in terms of hop counts) between failed flippers, candidate flippers are selected in following two ways. **Experiment 1.a:** The selected flippers are 1-hop away from each other. **Experiment 1.b:** The selected flippers are more than 2 hops distance apart. The chosen nodes are selected carefully, so that there exists at least one path between the source and the destination of each TCP flow even after the chosen nodes fails. The system can not accept a new flow, until the flipper readjustment converges. Therefore, the convergence time of the flipper readjustment is significant in case of failure. Convergence time for flipper readjustments are shown in Figure 5a. The results suggest that, the effect of multiple flipper failure is dependent on the separation of the failed flippers. However, the convergence time difference reduces for Experiments 1.a and 1.b at $k = 5$ as the diameter of the used topology is 5.

The role change of flippers results in path adjustment of flows. Out of generated 200 random flows, Figure 5a shows the number of flows needs to be readjusted due to the change in data plane topology. The plot signifies that the number of flow adjustments also depend on the separation between the failed flippers. The higher separation between failed flippers requires large number of role change operations to reach convergence. This results higher number of flow adjustment. The result also shows that, the increase in number of flipper failure increases the number of flows required to be rerouted.

C. Effect of Link Failure

To visualize the effect of link failure on data plane operation, we perform similar experiments as mentioned earlier. In this experimental setup, k links are chosen randomly so that there is at least one path between the source to the destination of each flow. We perform the following two experiments by selecting variable number of k links as following. **Experiment 2.a:** The failed links are 1-hop distance apart and **Experiment 2.b:** The failed links are at least 2-hop distance apart. These selected links are disconnected simultaneously to perform the failure experiments. The emulation results shown in Figure 5b reveals that, the convergence time and required number of flow adjustments depends on the number of failed links and separation between the failed links.

VII. Background and Related Works

Traditional SNMP based for network management system [25]–[27] resulted complex and rigid architectures. [28] shows that, network configurations are highly error prone. The error of configuration happens due to the complexity of managing each network devices individually. To reduce the network management overhead, SDN came into existence. Some of the popular SDN control plane approaches are, [29]–[33]. To ensure scalability, SDN control plane for service provider network needs to be distributed. According to Panda *et.al.* [34], it is not possible to ensure strong consistency, availability and partition tolerance simultaneously in case of distributed control platform. Increase in number of controllers increases scalability and management overhead both [35]. On the other hand, reduction of controllers makes the control plane a bottleneck [36]. Therefore, designing of distributed control platform for service provider network is non-trivial. Although, [7], [8] have proposed distributed control plane, fault-tolerance remains an issue in case of distributed control plane. To However, some of the fault resilient distributed control planes are refereed in [37]. Among the existing works, POCO-PLC [18] proposes a Pareto optimal, fault-resilient off-line control plane. However, designing a fault tolerant SDN network management system is non-trivial due to the fact that selecting a recovery strategy might take longer convergence time. These limitations have motivated us to design a dynamic architecture, which can reduce the flow initiation delay and can provide fault tolerance.

VIII. Commonly Asked Questions

Here we discuss the answer of some questions that may arise while reading this paper.

How FLIPPER is different from SDN?

Standard SDN platform uses static role assignments at the time of deployment. Static deployment limits performance of SDN in case of topology changing networks. In case of controller failure, SDN might cease to perform. In such cases, FLIPPER provides more availability than SDN by utilizing dynamic role assignment of flippers.

Can FLIPPER Work in fail-open and fail-close semantic?

Fail-open and fail-close semantics provide partition tolerance in case of fault resilient architecture. Fault-resilience architectures handle specific types of failures. In Section IV, we prove that, the proposed FLIPPER is fault-tolerant. In a fault-tolerant architecture the effect of failure only affects in terms of delay. Therefore, we argue that FLIPPER provides a stronger solution to handle network partitioning problem.

Why our emulation results are not comparable with existing works?

Existing SDN based architectures do not focus on fault tolerance, and most of the cases the solutions are off-line and static deployment based. So, they can not handle arbitrary failure. Once the SDN controllers fail, the switches under the influence of the controllers can not perform data forwarding until a new controller is configured to work with them. Therefore, the term convergence time becomes irrelevant in that context.

IX. Conclusion

In this work, we propose FLIPPER which supports SDN like network management and control, while avoiding the controller bottleneck problem, and supporting a stronger notion of fault tolerance. Built over the existing ONIX architecture, FLIPPER supports a scalable notion of dynamic role adaptation based on a distributed self-stabilizing algorithm. The simulation result shows the benefits of FLIPPER, whereas the emulation over a real testbed conveys the feasibility of FLIPPER implementation over the existing network infrastructure.

X. Acknowledgement

This work is supported by TATA Consultancy Services, India through TCS Research Fellowship Program. We would like to thank Dr. Sushanta Karmakar, for providing help in the mathematical proofs. We also thank anonymous reviewers for improving the presentation of the paper.

References

[1] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, vol. 103, no. 1, pp. 14–76, 2015.

[2] Z. A. Qazi, C.-C. Tu, L. Chiang, R. Miao, V. Sekar, and M. Yu, "SIMPLE-fying middlebox policy enforcement using SDN," *ACM SIGCOMM computer communication review*, vol. 43, no. 4, pp. 27–38, 2013.

[3] R. Ozdag, "Intel® ethernet switch FM6000 series-software defined networking," *See goo.gl/AnvOvX*, p. 5, 2012.

[4] S. Sezer, S. Scott-Hayward, P. K. Chouhan, B. Fraser, D. Lake, J. Finnegan, N. Viljoen, M. Miller, and N. Rao, "Are we ready for SDN? implementation challenges for software-defined networks," *IEEE Communications Magazine*, vol. 51, no. 7, pp. 36–43, 2013.

[5] D. Levin, M. Canini, S. Schmid, and A. Feldmann, "Incremental SDN deployment in enterprise networks," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 473–474.

[6] D. Levin, M. Canini, S. Schmid, F. Schaffert, and A. Feldmann, "Panopticon: Reaping the benefits of incremental SDN deployment in enterprise networks," in *Proceedings of 2014 USENIX Annual Technical Conference (USENIX ATC 14)*, 2014, pp. 333–345.

[7] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX Conference on OSDI, 2010*. USENIX Association, 2010, pp. 1–6.

[8] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the 3rd HotSDN, 2014*. ACM, 2014, pp. 1–6.

[9] N. Foster, R. Harrison, M. J. Freedman, C. Monsanto, J. Rexford, A. Story, and D. Walker, "Frenetic: A network programming language," in *ACM Sigplan Notices*, vol. 46, no. 9. ACM, 2011, pp. 279–291.

[10] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. Kompella, "Towards an elastic distributed SDN controller," in *ACM SIGCOMM Computer Communication Review*, vol. 43, no. 4. ACM, 2013, pp. 7–12.

[11] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," *IEEE Communications Letters*, vol. 18, no. 8, pp. 1339–1342, 2014.

[12] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed multi-domain SDN controllers," in *Proceedings of 2014 IEEE Network Operations and Management Symposium (NOMS)*. IEEE, 2014, pp. 1–4.

[13] F. Hu, Q. Hao, and K. Bao, "A survey on software-defined network and OpenFlow: From concept to implementation," *IEEE Communications Surveys Tutorials*, vol. 16, no. 4, pp. 2181–2206, 2014.

[14] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, and J. Turner, "Openflow: enabling innovation in campus networks," *ACM SIGCOMM Computer Communication Review*, vol. 38, no. 2, pp. 69–74, 2008.

[15] E. W. Dijkstra, "Self-stabilizing systems in spite of distributed control," *Communications of the ACM*, vol. 17, no. 11, pp. 643–644, 1974.

[16] V. Turau, "Linear self-stabilizing algorithms for the independent and dominating set problems using an unfair distributed scheduler," *Information Processing Letters*, vol. 103, no. 3, pp. 88–93, 2007.

[17] M. Gradinariu, S. Tixeuil *et al.*, "Self-stabilizing vertex coloring of arbitrary graphs," *Proceedings of OPODIS 2000*, pp. 55–70, 2000.

[18] D. Hock, M. Hartmann, S. Gebert, T. Zinner, and P. Tran-Gia, "Poco-plc: Enabling dynamic pareto-optimal resilient controller placement in sdn networks," *Proceedings of the 33rd INFOCOM, 2014*, 2014.

[19] "Ns-3.22 - nsnam," https://www.nsnam.org/wiki/Ns-3.22.

[20] "Snap autonomous systems AS-733 data set," http://snap.stanford.edu/data/as.html.

[21] "Snap autonomous systems - oregon-1 data set," http://snap.stanford.edu/data/oregon1.html.

[22] B. Lantz, B. Heller, and N. McKeown, "A network in a laptop: Rapid prototyping for software-defined networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hotnets,2010*. ACM, 2010, pp. 19:1–19:6.

[23] OVS, "Open vSwitch," http://openvswitch.org/.

[24] "Open vSwitch database server," http://openvswitch.org/support/dist-docs/ovsdb-server.1.txt.

[25] T. Benson, A. Akella, and D. A. Maltz, "Unraveling the complexity of network management." in *NSDI*, 2009, pp. 335–348.

[26] H. Ballani and P. Francis, "Conman: a step towards network manageability," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 205–216.

[27] X. Chen, Z. M. Mao, and J. van der Merwe, "Towards automated network management: Network operations using dynamic views," in *Proceedings of the 2007 SIGCOMM Workshop on Internet Network Management*, ser. INM '07. ACM, 2007, pp. 242–247.

[28] A. Bierman, M. Bjorklund, K. Watsen, and R. Fernando, "Restconf protocol," *IETF draft, work in progress*, 2014.

[29] M. Casado, T. Garfinkel, A. Akella, M. J. Freedman, D. Boneh, N. McKeown, and S. Shenker, "Sane: A protection architecture for enterprise networks." in *Usenix Security*, 2006.

[30] M. Casado, M. J. Freedman, J. Pettit, J. Luo, N. McKeown, and S. Shenker, "Ethane: Taking control of the enterprise," in *ACM SIGCOMM Computer Communication Review*, vol. 37, no. 4. ACM, 2007, pp. 1–12.

[31] M. Desai and T. Nandagopal, "Coping with link failures in centralized control plane architectures," in *Proceedings of the 2nd COMSNET, 2010*. IEEE Press, 2010, pp. 79–88.

[32] R. Niranjan Mysore, A. Pamboris, N. Farrington, N. Huang, P. Miri, S. Radhakrishnan, V. Subramanya, and A. Vahdat, "Portland: A scalable fault-tolerant layer 2 data center network fabric," *ACM SIGCOMM Computer Communication Review*, vol. 39, no. 4, pp. 39–50, 2009.

[33] Z. Yu, M. Li, X. Yang, and X. Li, "Palantir: Reseizing network proximity in large-scale distributed computing frameworks using sdn," in *Proceedings of the 7th CLOUD, 2014*. IEEE, 2014, pp. 440–447.

[34] A. Panda, C. Scott, A. Ghodsi, T. Koponen, and S. Shenker, "Cap for networks," in *Proceedings of the 2nd HotSDN, 2013*. ACM, 2013, pp. 91–96.

[35] D. Levin, A. Wundsam, B. Heller, N. Handigol, and A. Feldmann, "Logically centralized?: state distribution trade-offs in software defined networks," in *Proceedings of the 1st HotSDN,2012*. ACM, 2012, pp. 1–6.

[36] S. H. Yeganeh, A. Tootoonchian, and Y. Ganjali, "On scalability of software-defined networking," *IEEE Communications Magazine,*, vol. 51, no. 2, pp. 136–141, 2013.

[37] A. S. da Silva, P. Smith, A. Mauthe, and A. Schaeffer-Filho, "Resilience support in software-defined networking: A survey," *Computer Networks*, vol. 92, Part 1, pp. 189 – 207, 2015.

Primary Path Effect in Multi-Path TCP: How Serious Is It for Deployment Consideration?

Subhrendu Chattopadhyay
Sukumar Nandi
IIT Guwahati, India 781039
subhrendu@iitg.ernet.in
sukumar@iitg.ernet.in

Samar Shailendra
TCS Research & Innovation
Bangalore, India 560100
s.samar@tcs.com

Sandip Chakraborty
IIT Kharagpur
Kharagpur, India 721302
sandipc@cse.iitkgp.ernet.in

ABSTRACT

This poster provides an in-depth analysis of the primary path effect in Multi-path TCP using thorough experimentation over a realistic network setup. We observe the impact of various primary path parameters, like bandwidth, delay and loss, over the end-to-end performance. It is shown that under certain circumstances overall network performance can be improved by more than 50% with proper primary path selection. This study may drive the research community towards the design of new segment scheduling algorithms considering the effect of primary path selection over Multi-path TCP.

CCS CONCEPTS

• **Networks** → *Transport protocols*;

KEYWORDS

MPTCP, Primary Path Effect, Performance

ACM Reference format:
Subhrendu Chattopadhyay, Sukumar Nandi, Samar Shailendra, and Sandip Chakraborty. 2017. Primary Path Effect in Multi-Path TCP: How Serious Is It for Deployment Consideration?. In *Proceedings of Mobihoc '17, Chennai, India, July 10-14, 2017,* 2 pages.
DOI: http://dx.doi.org/10.1145/3084041.3084076

1 INTRODUCTION

With the advent of multi-interface communication supports over personal computing devices like smartphones and tablets, the concept of multi-path protocols have gradually become popular for large scale deployments. The networking community has started exploring pros and cons of various aspects of multi-path support over the popular and widely accepted transmission control protocol (TCP), called Multi-path TCP (MPTCP) [3], that uses multiple end-to-end paths through multiple interfaces attached with the end hosts. MPTCP initiates an end-to-end connection using three way handshaking with MP_CAPABLE flag, through one of the available interfaces known as the *primary path*. Once the end-to-end connection is established over the primary path, MPTCP explores the alternate paths, and appends those *secondary paths* to the main flow to aggregate available bandwidth over multiple paths.

Although a few existing works [1, 2] have observed that the overall MPTCP performance depends on the selection of primary path, however, there is no concrete study on how various primary path parameters, like bandwidth, delay and packet loss, impact end-to-end transport performance. In this poster, we explore scenarios when primary path selection can be a serious issue that need to be considered while designing the path scheduler. We discuss our observations from thorough experiments over a realistic MPTCP setup, where we measure the effect of delay, bandwidth and loss rate of primary path over the MPTCP performance. We observe that under certain scenarios, MPTCP performance can be improved as high as around 60%, if the primary path is chosen properly. This study provides a strong motivation for driving the research community towards the development of path selection algorithms based on the end-to-end path properties.

2 EXPERIMENTAL ANALYSIS OF PRIMARY PATH EFFECT

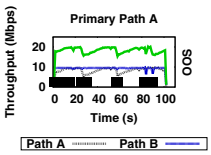
The experimental setup has been built over Mininet network emulator platform where two hosts H1 and H2 are connected via two paths, Path A and Path B, through two different interfaces at each host. The end hosts as well as all the network switches in the path use Linux based operating system, and we use Linux tool `iperf` to generate the network traffic. The Linux kernel at the hosts are configured with MPTCP V0.90[1]. All the experiments are carried out for two cases – one by selecting Path A as the primary path and the other with Path B as the primary path. The bandwidth, delay and loss rate for Path B is kept constant at 10 Mbps, 15 ms and 0%, respectively. We have carried out three experiments by varying the parameters {bandwidth, delay, loss rate} for Path A – (a) **Exp 1 (delay difference)**: {10$Mbps$, 250ms, 0%}, (b) **Exp 2 (bandwidth difference)**: {5$Mbps$, 15ms, 0%}, and (c) **Exp 3 (loss rate difference)**: {10$Mbps$, 15ms, .5%}.

2.1 Effect on Transport Layer Throughput

Fig. 1 represents the difference in throughput when there exists a significant delay difference between the primary path and secondary path. The overall throughput reduces significantly in case of Primary Path A (slower path). The results also show that, the number of out of order segments (OOS) generated are significantly higher in case of slower primary path (i.e. Path A). MPTCP retransmission due to OOS is handled by resending the segments

[1] https://www.multipath-tcp.org

Mobihoc '17, Chennai, India
 978-1-4503-4912-3/17/07…$15.00
DOI: http://dx.doi.org/10.1145/3084041.3084076



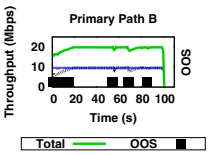
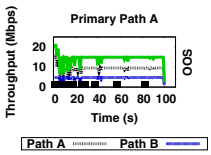


Figure 1: Effect of Delay (Exp 1)



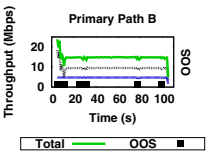
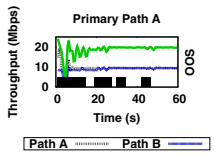


Figure 2: Effect of Bandwidth (Exp 2)



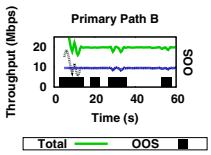


Figure 3: Effect of Loss Rate (Exp 3)

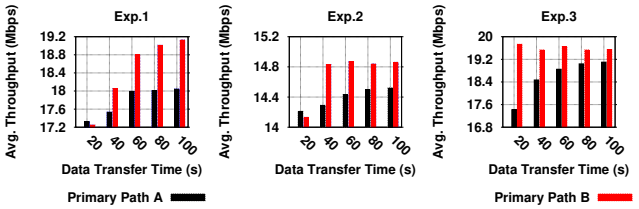


Figure 4: Effect of Flow Duration

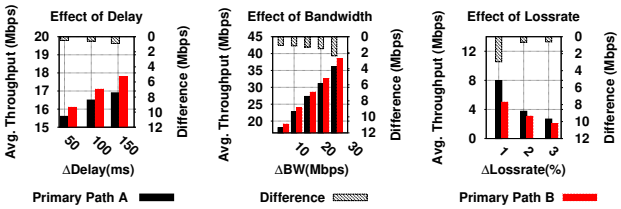


Figure 5: Throughput variation

via the same path for two times, and after that, the segment is assigned to a different path. A retransmission due to three duplicate acknowledgement affects severely in case when the primary path is Path A. The primary path takes longer time to converge to the highest attainable bandwidth due to higher round trip time (RTT). Retransmission in primary path due to OOS further worsen the convergence. This phenomenon is absent in the case when the primary path is Path B, as generation of OOS can be quickly mitigated by retransmitting the segment via that path due to less end-to-end delay. Due to this behavior, the selection of primary path is more significant in case of delay deference, even for longer flows. Fig. 4 shows the impact of flow duration on the MPTCP performance for the three experimental scenarios. As flow duration increases, delay difference has more impact on the throughput difference based on the primary path selection.

In case of a bandwidth difference in between the primary and the secondary path, we see a similar effect (Fig. 2). This result also suggests that choosing a higher bandwidth path as the primary path can significantly reduce the generation of OOS. Fig. 3 shows the results when primary path has higher loss rate than that of secondary path. The results suggests that, choosing a high loss rate path as the primary path can reduce the overall throughput. Further from Fig. 4, we observe that the impacts of bandwidth difference and loss rate difference are more in case of short duration flows.

2.2 Impact of Parametric Difference Between Two Paths

Next we analyze how the delay, bandwidth and throughput difference between the two paths impact the MPTCP throughput based on primary path selection. Fig. 5 represents the change in aggregated throughput with the change in delay, bandwidth and loss rate difference between the two paths. From these figures, it is clear that the increase in variability between the two paths have significant impact on the throughput performance based on the selection of primary path. The impact is significant under certain cases. For example, in case of loss rate (Exp 3), selecting a low loss rate path as the primary path can improve the MPTCP throughput as high as 60%, as we can observe from Fig. 5.

2.3 Summary of Observations

From the experimental results, we observe that a primary path with lower bandwidth, higher delay and/or lower loss rate gives rise to OOS at the receiver (Fig. 1, Fig. 2 and Fig. 3). Such an increase in OOS increases the number of triple-duplicate ACKs at the sender causing the reduction in the congestion window. This unduly reduction in congestion windows adversely affects the aggregated throughput of the network. The main reason behind the increase in OOS is, traditional RTT based congestion control algorithms are not suitable for disparate path characteristics between sender and the receiver.

It can also be observed from Fig. 4 that the effect of delay disparity between paths is more detrimental than the effect of bandwidth disparity between paths. This is due to the very nature of congestion control algorithm and its direct dependence over the RTT. Moreover, the effect of path selection is visible for both the short flows as well as the long flows.

3 CONCLUSION AND NEXT STEPS

In this poster, we have raised a serious issue on MPTCP performance considerations that need to be addressed properly for its practical deployment scenarios. For smartphone with multi-interface (say cellular and Wi-Fi) support, selecting the cellular interface in a poor connectivity region for setting up the primary path may adversely effect the MPTCP performance. Our study gives a few preliminary insights of the primary path effect on MPTCP performance, which can drive the research community to dig into the details of this phenomenal problem.

REFERENCES

[1] Q. De Coninck, M. Baerts, B. Hesmans, and O. Bonaventure. 2015. Poster: Evaluating Android Applications with Multipath TCP. In *21st MobiCom*. ACM, 230–232.
[2] B. Sonkoly, F. Németh, L. Csikor, L. Gulyás, and A. Gulyás. 2014. SDN based Testbeds for Evaluating and Promoting Multipath TCP. In *IEEE ICC*. 3044–3050.
[3] C. Xu, J. Zhao, and G. M. Muntean. 2016. Congestion Control Design for Multipath Transport Protocols: A Survey. *IEEE Comm. Surveys & Tutorials* 18, 4 (Fourthquarter 2016), 2948–2969.

Improving MPTCP Performance by Enabling Sub-Flow Selection over a SDN Supported Network

Subhrendu Chattopadhyay
IIT Guwahati
Guwahati, India 781039
subhrendu@iitg.ernet.in

Samar Shailendra
TCS Research & Innovation
Bangalore, India 560100
s.samar@tcs.com

Sukumar Nandi
IIT Guwahati
Guwahati, India 781039
sukumar@iitg.ernet.in

Sandip Chakraborty
IIT Kharagpur
Kharagpur, India 721302
sandipc@cse.iitkgp.ernet.in

***Abstract*—The primary objective behind the development of Multipath TCP (MPTCP) is to aggregate throughput by creating multiple sub-flows via different network interfaces. A difference in end-to-end path characteristics for the sub-flows may generate out of order segments, causing head of line (HOL) blocking at the receiver. An intelligent selection of a subset of the available sub-flows can reduce the number of out of order segments; thus sub-flow selection can enhance the performance of MPTCP. In this paper, we first propose a Markov model for the performance of MPTCP in terms of end-to-end sub-flow characteristics. Based on the theoretical model, we present an optimization framework for active sub-flow selection by exploiting the controller functionalities over a software defined network (SDN) architecture. Finally, experimental results are obtained to demonstrate performance improvements of MPTCP in terms of aggregated throughput.**

***Keywords*-MPTCP, Sub-flow selection, SDN**

I. Introduction

Modern day devices are usually equipped with multiple hardware interfaces that can be leveraged to satisfy the demand for increasing traffic by aggregating the available bandwidth at all interfaces. *Multipath Transmission Control Protocol* (MPTCP) [1] has been proposed in the literature as an end-to-end protocol for data-center and enterprise networks with the availability of multi-interface networking devices, which provides the support for bandwidth aggregation via concurrent usage of different interfaces by creating multiple sub-sockets. MPTCP initiates multiple sub-sockets via different interfaces to aggregate the bandwidth.

The current Linux kernel implementation of MPTCP [2] consists of three major modules: *Path Manager*, *Segment Scheduler*, and *Congestion Control Mechanism*. The *Path Manager* module manages the available sub-flows between the end hosts. Currently, MPTCP has proposed two choices of path manager. **(a) Full-mesh path manager** creates sub-sockets for between all available pair of interfaces. On the other hand, **(b) ndiffports** selects k sub-flows among all available sub-flows, where k is a user defined parameter. MPTCP *Congestion Control* module manages congestion window for each sub-flow separately. Several congestion control algorithms like *Linked Increase Algorithm* (LIA) [3], *Opportunistic Linked Increase Algorithm* (OLIA) [4], *Balanced Linked Increase Algorithm* (BALIA) [5] etc. [6], [7] have been proposed for MPTCP. Once the congestion window size for each path is decided, segment scheduler takes the responsibility of scheduling the segments for the individual sub-flows. *Round-Robin* and *lowest RTT First* are the two available segment scheduling strategies described in the MPTCP standard.

The primary task of a segment scheduler is to reduce out of order packets at the receiver. However, in a network, the path characteristics (such as bandwidth, delay, loss rate, jitter etc.) of the underlying sub-flows can be significantly different as well as time varying. The differences in end-to-end path characteristics of each active sub-flow may lead to an increase in out of order segments delivered at the receiver. In [8], the authors have tried to limit the receiver buffer in order to decrease out of order segment delivery by employing network coding. However, it has been found that, their implementation violate MPTCP basic principle of do no harm objective [9]. On the other hand, [10] and [11] focuses on the segment scheduling mechanism to avoid out of order segment generation. However, segment scheduling alone can not reduce out of order delivery and may lead to *Head of Line (HOL) blocking* at the receiver side [12]. HOL blocking increases delays and packet drops. Thus, the number of spurious retransmission also increases. Therefore, Cao *et.al.* [12] proposes a receiver buffer aware path selection mechanism. However, like most of the transport layer protocols, their implementation uses round trip time (RTT) as a measure of path characteristics. In case of MPTCP, one segment and its acknowledgment might follow different paths. So, RTT is not a faithful estimate of a path at the sender side. Therefore, relying on simple RTT driven path characteristics leads to severe performance degradation in MPTCP performance. In our previous work [13] we have shown that MPTCP provides near optimal experience, when the active sub-flows have similar path characteristics, as in those cases RTT provides a good estimation. However, the difference in delay, effective bandwidth, and loss rate can significantly increase the number of out of order segments at the receiver [14]. Therefore, we argue instead of relying on the RTT, MPTCP must rely on end to end path characteristics. Based on the end to end semantics, MPTCP path management module must choose a set of sub-flows which can avoid HOL blocking by reducing out of order delivery [15].

Therefore, in this paper, we propose a *Software Defined Networking* (SDN) [16] aided intelligent dynamic path management scheme. SDN provides a logically centralized view of network topology parameters to the application protocols

978-1-5386-6876-4/18/$31.00 ©2018 IEEE

by periodically obtaining statistics from all its data plane devices [17]. This makes it feasible to optimize the end-to-end performance of MPTCP by selecting a suitable active set of MPTCP sub-flows. We consider an enterprise or data-center network, where the network switches are connected with a SDN controller that can estimate sub-flow characteristics based on end-to-end path properties. As SDN cannot obtain the information about receiver buffer evolution as well as the prediction of aggregated MPTCP throughput based on the sub-flow properties, building up a SDN aided path manager application is non-trivial. Therefore, in this paper, we propose an estimation mechanism to predict the MPTCP aggregated throughput for a set of sub-flows with their end-to-end path characteristics (latency, available bandwidth, etc.). Unlike prior works our proposed model provides aggregated throughput for a given set of sub-flows. Consequently, we take a two-stage approach in this paper. We formulate an irreducible and aperiodic *Discrete Time Markov Chain* (DTMC) to model the aggregated throughput prediction of a MPTCP flow with the end-to-end path characteristics of a given set of sub-flows (Section III). Based on the predicted throughput from the estimator model, we develop an optimization framework to find out the optimal set of sub-flows that can maximize the aggregated throughput for a given MPTCP flow (Section IV). The SDN controller executes this optimization framework and schedules the sub-flows accordingly. Finally, we evaluate the performance of the proposed mechanism and compare it with various baselines.

II. Network and System Model

The objective of this paper is to develop a solution for dynamic sub-flow management while considering the end-to-end path characteristics. The problem is to identify a set of sub-flows from all the available paths between a source-destination pair of a MPTCP flow, such that (i) the overall MPTCP aggregated throughput is maximized, and (ii) the receiver buffer size is always limited by a certain threshold to avoid HOL blocking problem. However, obtaining sub-flow characteristics (like receiver buffer evolution) under a dynamic scenario is non-trivial over a complete distributed network management framework, and therefore we leverage on SDN based network management concept in this paper. We consider an enterprise or data-center network, where the network switches are connected with a SDN controller that can estimate sub-flow characteristics based on end-to-end path properties.

Although a SDN controller can monitor end-to-end path characteristics like latency and available bandwidth, obtaining the information about receiver buffer evolution as well as the prediction of aggregated MPTCP throughput based on the sub-flow properties are non-trivial. Therefore, we need to build up an estimation mechanism to predict the MPTCP aggregated throughput for a set of sub-flows with their end-to-end path characteristics (latency, available bandwidth etc.). To the best of our knowledge, the prior works on MPTCP do not model aggregated throughput for a given set of sub-flows. Consequently, we take two-stage approach in this paper as follows.

1) We formulate an irreducible and aperiodic *Discrete Time Markov Chain* (DTMC) to model the aggregated throughput prediction of a MPTCP flow with the end-to-end path characteristics of a given set of sub-flows (Section III).
2) Based on the predicted throughput from the estimator model, we develop an optimization framework to find out the optimal set of sub-flows that can maximize the aggregated throughput for a given MPTCP flow (Section IV). The SDN controller executes this optimization framework and schedules the sub-flows accordingly.

A. Network and System Model

We assume a network as a undirected graph $G = \{V, E\}$, where the vertices represent network switches and hosts, and the edges represent physical connectivity between them. Let $\mathcal{S}$ be the set of all node disjoint sub-flows between a sender and the corresponding receiver. A MPTCP flow is a collection of sub-flows; therefore, we represent $\mathcal{S} = \{S_1, S_2 \dots S_n\}$, where S_k represents k^{th} sub-flow, and n represents the total number of node-disjoint sub-flows between the corresponding sender-receiver pair. A sub-flow $S_k = \{v_1^k, v_2^k \dots v_{n_k}^k\}$ is equivalent to an ordered set of vertices, where each $v_i^k \in V$ such that $v_1^k, v_2^k \dots v_{n_k}^k$ forms a path of hop count of n_k between the sender-receiver pair. As a consequence, we use the terms "path" and "sub-flow" interchangeably, where "sub-flow" represents a MPTCP connection whereas "path" indicates the underlying network path between the sender and the receiver. Let $e_{ij}^k \in E$ denotes an edge between the two nodes v_i^k and v_j^k. Let B_{ij}^k and L_{ij}^k represent the bandwidth and loss rate of e_{ij}^k. We further assume that the propagation and queueing delay of e_{ij}^k follow independent normal distribution with mean D_{ij}^k and standard deviation Θ_{ij}^k.

We consider that the end-to-end path characteristics of S_i can be represented by following three tuples: $q_i = \{b_i, Pr_i(X = r), l_i\}$, where b_i and l_i represent the bandwidth and the segment loss probability of S_i, respectively. Note that throughout the paper, we use the terms packet and segment interchangeably. $Pr_i(X = r)$ represents the probability mass function (pmf) for RTT of S_i being r. For the sake of simplicity we assume that, $\forall i : Pr_i(X = r)$ follows independent truncated normal distribution with mean μ_i and standard deviation σ_i. Therefore, $Pr_i(X = r) = \Psi(\mu_i, \sigma_i, 0, \infty; X = r)$ where $\Psi(X = r; \mu, \sigma, a, b)$ represents the cumulative probability density function of a random variable X having mean as μ and standard deviation σ such that $\forall X : a \leq X \leq b$. By using addition rule of normal distribution, we get $\mu_i \approx 2\sum_{jk}(D_{j,k}^i)$ and $\sigma_i^2 \approx 2\sum_{j,k}(\Theta_{jk}^i)^2$. For the sake of simplicity we use the notation $Q_i = \{b_i, l_i, \mu_i, \sigma_i\}$. Here, Q_i signifies the path characteristics of S_i. For ease of representation we use $\vec{Q} = \{Q_i\}$.

Each sub-flow maintains a separate congestion window. The size of congestion window of S_i at time t is represented as

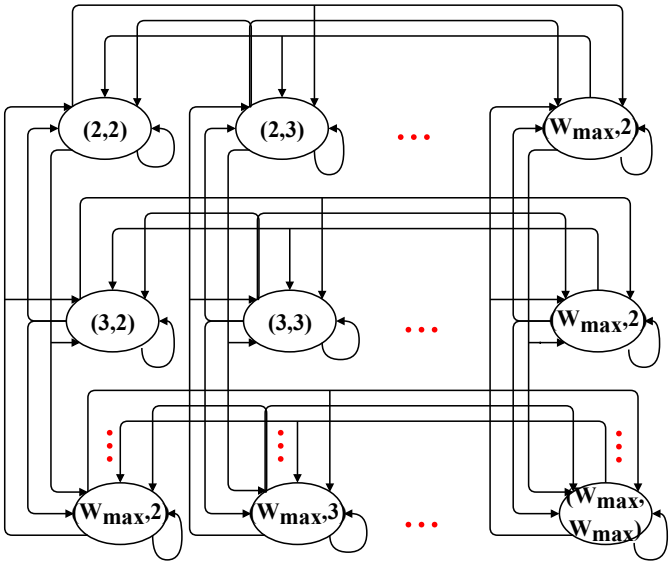


Fig. 1: Markov Model for a MPTCP with 2 Sub-Flows

$w_i(t)$. We use $\mathcal{T}$ and $\mathcal{R}$ to signify the steady state throughput and the receiver buffer size of the MPTCP connection between the intended sender-receiver pair. Our objective is to estimate the value of $\mathcal{T}$ and $\mathcal{R}$ in terms of Q_i that represents the underlying path characteristics of the MPTCP sub-flows for a sender-receiver pair.

III. Impact of MPTCP Sub-flow Selection on Throughput Performance – An Estimation Model

In this section, we use an irreducible and aperiodic DTMC model to estimate the values of aggregated steady state throughput ($\mathcal{T}$) and the receiver buffer size ($\mathcal{R}$) based on the path characteristics q_i. Considering n number of possible sub-flows $\{S_1, S_2, ..., S_n\}$, the states of a MPTCP flow can be represented through the congestion window across those n different sub-flows. Therefore, a state in the system can be represented as $\{w_1, w_2, ..., w_n\}$ where w_i is the congestion window value of the sub-flow S_i. We make the following assumptions,

- The change in congestion window of a path is triggered based on a discrete event system by observing the corresponding RTT of the underlying path.
- The congestion window updates at different paths are mutually independent and identically distributed.

Therefore, the congestion window evolution of i^{th} sub-flow of a MPTCP flow can be represented as a stochastic process $w_i(t)$. Accordingly, we develop a n dimensional irreducible and aperiodic discrete time Markov model, where a state of the system is represented by n-tuple $\{w_1, w_2, ..., w_n\}$, where each $w_i \in [2, W_{max}]$, W_{max} being the maximum congestion widow value. An example DTMC for $n = 2$ is shown in Figure 1. The state transition is allowed when a segment is either received successfully or it is lost in the transmission. Transition triggering events are handled on a sub-flow level. Therefore, we assume that state transition triggered by sub-flow k alters only the k^{th}-element of the state variable. We term this property of our model as *"single path transition"* property. Let us denote the state transition probability from state $(w_1, ..., w_r, ...w_n)$ to $(w_1, ..., w'_r, ...w_n)$ by $P_{(w_1,...,w_r,...w_n);(w_1,...,w'_r,...w_n)}$. Without any loss in generality we use the notation $P_{(w_r;w'_r)}$ to indicate the transition probability $P_{(w_1,...,w_r,...w_n);(w_1,...,w'_r,...w_n)}$. All the popular MPTCP congestion control algorithm (like BALIA [18]) adapts the congestion window size of a sub-flow based on the RTT estimation along that sub-flow. Therefore the state transition probabilities of the proposed DTMC depend on the RTT estimation. At this stage we ask this question: *What can be the RTT estimate (r_i) at path (sub-flow) S_i, that can trigger a change of congestion window size to w'_i from w_i?*

A. Estimation of RTT for Congestion Window Size Adaptation

Although any MPTCP congestion control algorithm can be used for our modelling purpose, we use "BALIA" [18] as a representative case. As shown in [5], BALIA congestion control algorithm can be represented using the family of equations given by Eq. (1a) (for successful segment transmission) and Eq. (1b) (for a transmission failure), where $Y_i(t) = \frac{w_i(t)}{r_i}$ and $\alpha_i(t) = \frac{\max_k\{Y_k(t)\}}{Y_i(t)}$. In this case, r_i represents the measured RTT of S_i.

$$w'_i = \begin{cases} \frac{Y_i(t)}{r_i\left(\sum_k Y_k(t)\right)^2}\left(\frac{1+\alpha_i(t)}{2}\right)\left(\frac{4+\alpha_i(t)}{5}\right) & \text{Success (1a)} \\ \frac{w_i(t)}{2}\min\{\alpha_i(t), 1.5\} & \text{Failure (1b)} \end{cases}$$

Based on above estimation of the congestion window size as given for BALIA congestion control algorithm, our objective is to find out r_i that can trigger a congestion window size of w'_i.

Let $\sum_k Y_k(t) = (C_{-i}(t) + Y_i(t))$ and $\max_k\{Y_k(t)\} = Y_m(t)$. Therefore, $\alpha_i(t) = \frac{w_m(t)r_i}{r_m w_i(t)}$. So, Eq. (1a) simplifies to Eq. (2a) and Eq. (1b) reduces to Eq. (2b). From this point onwards, we use w_i,w'_i and C_{-i} instead of $w_i(t)$, $w_i(t+1)$ and $C_{-i}(t)$ for notational simplicity.

$$w'_i = \begin{cases} \frac{w_i}{r_i^2\left(C_{-i}+\frac{w_i}{r_i}\right)^2}\left(\frac{4r_m^2w_i^2+5r_iw_iw_mr_m+r_i^2w_m^2}{10r_m^2w_i^2}\right) & \text{(2a)} \\ \frac{w_i}{2}\min\{\frac{w_mr_i}{r_mw_i}, 1.5\} & \text{(2b)} \end{cases}$$

By solving Eq. (2), we get

$$r_i = \frac{5}{4}\left(\frac{-w_mw_ir_m}{2r_m^2w_iw'_i} + C_{-i}w_i\right) \pm \sqrt{\left(\left(\frac{w_mw_ir_m}{2r_m^2w_iw'_i} - 2C_{-i}w_i\right)^2 - \frac{8}{5}\left(\frac{w_m^2}{10r_m^2w'_iw_i} - C_{-i}^2\right)\right)} \quad (3)$$

Let $\vec{W} = \{w_1, w_2, ...w_n\}$ and $\vec{R} = \{r_1, r_2, ..., r_n\}$. From Eq. (3), we can observe that r_i is a function of $\vec{W}$, $\vec{R}$ and m. Note that here S_m is the path for which $Y_k(t)$ is maximum. we denote $r_i = f(m, \vec{W}, \vec{R})$ where $f(.)$ is the corresponding function as given in Eq. (3). We consider two cases – (i) Y_k is maximum for the current path S_i under consideration ($\max\{Y_k\} = Y_i$, and $m = i$), and (ii) Y_k is maximum for

some other path S_m such that $m \neq i$. Therefore,

$$r_i = \begin{cases} f(i, \vec{W}, \vec{R}) & \text{if } \max\{Y_k\} = Y_i \\ f(m, \vec{W}, \vec{R}) & \text{otherwise} \end{cases} \quad (4)$$

By substituting $m = i$ in Eq. (3), we derive Eq. (5).

$$f(i, \vec{W}, \vec{R}) = \frac{w_i \pm \sqrt{w_i^2 + \frac{12w_i}{5w_i'} + 1.6}}{2C_{-i}} \quad (5)$$

Similarly, Eq. (2b) can be simplified also as follows.

$$r_i = \begin{cases} \dfrac{2w_i' r_m}{w_m} & w_i' \leq \dfrac{3w_i}{4} \text{ and } \max\{Y_k\} = Y_m \quad (6a) \\ 0 \geq r_i < \infty & \text{Otherwise} \quad (6b) \end{cases}$$

Now we can argue that given $\vec{W}$ and $\vec{R}$, the required RTT r_i can be calculated as per Eq. (4), Eq. (6a) and Eq. (6b). Therefore, we proceed for estimating the state transition probabilities of the proposed DTMC based on this RTT estimation.

B. Estimation of State Transition Probabilities

According to Eq. (4) and Eq. (6), transition events are:

1) SS_i**:** If the segment is delivered successfully via S_i, there can be two possible cases as follows:
   a) SS_{max_i}**:** This transition event is triggered if $m = i$, that is $\max\{Y_k\} = Y_i$ for path S_i after the successful delivery. As per the definition of Y_k, $Y_k \propto \frac{1}{r_i}$. Therefore, $\max\{Y_k\} = Y_i$ represents $\min\{r_k\} = r_i$.
   b) SS_{max_m}**:** If $m \neq i$, then $\exists m \in \{1, 2, ..., i-1, i+1, ..., n\} : \max\{Y_k\} = Y_m$. This event is the complement event of SS_{max_i}.
2) SL_i**:**If the segment is delivered successfully via S_i, there can be two possible cases as follows:
   a) SL_{max_i}**:** This transition event is triggered if there is a segment loss reported, and $\max\{Y_k\} = Y_i$. In this case, according to Eq. (6), the value of this event does not depend on r_i. Therefore, we consider $0 \geq r_i \geq \infty$. In such cases the only allowed sub-event is $w_i' = \frac{3w_i}{4}$. To signify this event, we use an indicator variable $\Gamma(w_i, w_i')$ as given in Eq. (7).

$$\Gamma(w_i, w_i') = \begin{cases} 1 & \text{if } 4w_i' = 3w_i \\ 0 & \text{Otherwise} \end{cases} \quad (7)$$

   b) SL_{max_m}**:** If $m \neq i$, then $\exists m \in \{1, 2, ..., i-1, i+1, ..., n\} : \max\{Y_k\} = Y_m$. This event is the complement event of SS_{max_i}. Whenever this event is triggered, transition of $w_i' > \frac{3w_i}{4}$, becomes impossible (see, Eq. (6)). Therefore, we only consider here the sub-event $w_i' \leq \frac{3w_i}{4}$. To notify this sub-event, we use a separate indicator variable $\Delta(w_i, w_i')$ as given in Eq. (8).

$$\Delta(w_i, w_i') = \begin{cases} 1 & \text{if } 4w_i' \leq 3w_i \\ 0 & \text{Otherwise} \end{cases} \quad (8)$$

Now from the above set of events, we can say $pr(SL_i) = l_i$ and $pr(SS_i) = (1 - l_i)$, where $pr(E_i)$ denotes the probability of event E_i. Based on the set of events, we simplify the transition probability $P_{(w_i;w_i')}$ by repeatedly applying law of total probability as given in Eq. (9).

$$P_{(w_i;w_i')} = pr(SS_i)pr(w_i'|SS_i) + pr(SL_i)pr(w_i'|SL_i) \quad (9)$$

where,

$$pr(w_i'|SS_i) = pr(w_i'|SS_{max_i})pr(SS_{max_i}) + pr(w_i'|SS_{max_m})pr(SS_{max_m})$$

and,

$$pr(w_i'|SL_i) = \Gamma(w_i, w_i')pr(SL_{max_i}) + \Delta(w_i, w_i')pr(w_i'|SL_{max_m})pr(SL_{max_m})$$

It can be noted from Eq. (2) that with BALIA, new congestion window (w_i') should be less than or equals to $\frac{3}{4}$th of original congestion window (w_i) when a segment loss occurs. The indicator variable $\Gamma(w_i, w_i')$ ensures this and accordingly we compute $pr(w_i'|SL_i)$. Now both the events SS_{max_i} and SL_{max_i} are equivalent to the event of i^{th} sub-flow having minimum r_i. According to our conjecture, $\forall i : Pr_i(X = r)$ are independent and identically distributed. Therefore, $pr(SS_{max_i}) = pr(SL_{max_i}) = \mathcal{Z}$ reduces to Eq. (10).

$$\mathcal{Z} = \int_{r=0}^{\infty} Pr_i(X = r) \prod_{k \neq i} Pr_k(X < r) dr \quad (10)$$

Similarly, $pr(SS_{max_m}) = 1 - pr(SS_{max_i})$ and $pr(SL_{max_m}) = 1 - pr(SL_{max_i})$ and other conditional probabilities can be calculated as follows – (a) $pr(w_i'|SS_{max_i}) = pr(X < f(i, \vec{W}, \vec{R}))$, (b) $pr(w_i'|SS_{max_m}) = pr(X < f(m, \vec{W}, \vec{R}))$, and (c) $pr(w_i'|SL_{max_m}) = pr(X < \frac{2w_i' r_m}{w_m})$.

This way we obtain the transition probability from state $(w_1, w_2, ..., w_i, ..., w_n)$ to state $(w_1, w_2, ..., w_i', ..., w_n)$ ($P_{(w_i;w_i')}$) based on Eq. (9).

C. Estimation of Average MPTCP Throughput

We now compute the average throughput of a MPTCP flow considering the data transfer rate through all its sub-flows. Let us consider that, $\vec{\Pi} = \left[\pi_{(2,2,...2)}, \pi_{(2,2,...,2,3)}, \ldots, \pi_{(W_{max_1}, W_{max_2}, ..., W_{max_n})}\right]$ be the stationary probability distribution vector of the states for the given DTMC. Therefore, by using Markovian property, stationary distribution of this DTMC can be calculated as per the following system of equations.

$$\pi_{w_1,...,w_n} = \sum_{k_1=2}^{W_{max_1}} \pi_{k_1,...,w_n} P_{(k_1;w_1)} + \ldots + \sum_{k_n=2}^{W_{max_n}} \pi_{w_1,...,k_n} P_{(k_n;w_n)} \quad (11)$$

We also have the normalization equation from the DTMC, which can be represented as follows.

$$\sum_{w_1=2}^{W_{max_1}} \sum_{w_2=2}^{W_{max_2}} \dots \sum_{w_n=2}^{W_{max_n}} \pi_{w_1,w_2 \dots w_i,\dots,w_n} = 1 \tag{12}$$

Let us define a "round" as the interval between two successive state transition events. If the system is currently under state $(w_1, w_2, ..., w_n)$, then the total number of segments that can be sent is calculated as $\sum_{j=1}^{n} w_j$. Therefore, the average number of segments sent by a state $(w_1, w_2, ..., w_n)$ is $\pi_{(w_1,w_2,\dots,w_n)} \sum_{j=1}^{n} w_j$. Consequently, the average number of segments that can be sent in one round (denoted as $Avg_C(\vec{Q})$ for a given configuration $\vec{Q} = \{q_1, q_2, ..., q_n\}$) is expressed as Eq. (13).

$$Avg_C(\vec{Q}) = \sum_{\forall w_i} \left(\pi_{(w_1,w_2,\dots,w_n)} \sum_{j=1}^{n} w_j \right) \tag{13}$$

Now we have to compute the average time for a round. The average time for a round includes (a) total data transmission time (time to transmit $Avg_C(\vec{Q})$ number of segments), (b) the time to receive the acknowledgements for the transmitted segments, and (c) the time for retransmission of lost segments. We assume a x-duplicate acknowledgement scheme, where a segment is retransmitted if the sender receives x number of consecutive duplicate acknowledgements. Assume that the segment size is s_s and acknowledgement size is a_s. Then for a given $\vec{Q}$, the average time required for one round ($Avg_T(\vec{Q})$) is computed as follows.

$$\mathcal{G}(\vec{Q}) = \max_x \left\{ \frac{((w_s^{avg} + x) \times s_s)}{b_s} + \rho \frac{w_s^{avg} \times a_s}{b_s} \right\} \tag{14}$$

In this case w_s^{avg} represents the average number of segments sent by a MPTCP sub-flow S_s and ρ is the RTT of that sub-flow.

Therefore, using Eq. (13) and Eq. (14), the average throughput is calculated as,

$$Avg_{Th}(\vec{Q}) = \frac{Avg_C(\vec{Q})}{\mathcal{G}(\vec{Q})} \tag{15}$$

D. Estimation of Receiver Buffer Size

The receiver buffer occupancy increases mainly due to the out of order segment delivery. We define segment seg_i as a 'key' segment if all other segments $seg_j : j > i$ reach to the destination before seg_i. All the seg_j must wait at the receiver buffer for the key segment, in order to ensure reliable delivery. Therefore, the occupancy of receiver buffer depends on the event that seg_j is successfully delivered before seg_i. We denote seg_i^{max} as the segment which stays in the queue for the longest time for a key segment seg_i. So, the receiver buffer length (RL) can be expressed as Eq. (16), where $\Delta(seg_k, seg_l)$ denotes the arrival time difference between seg_k and seg_l.

$$RL = |\Delta(seg_i^{max}, seg_i)| \times \text{throughput} \tag{16}$$

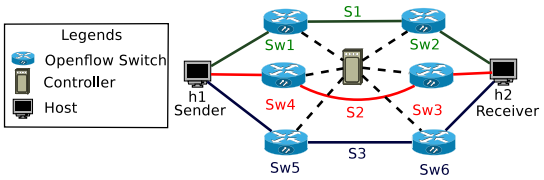


Fig. 2: Topology Structure for Experiments

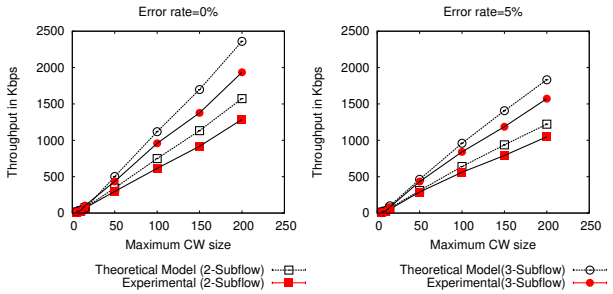


Fig. 3: Throughput Comparison

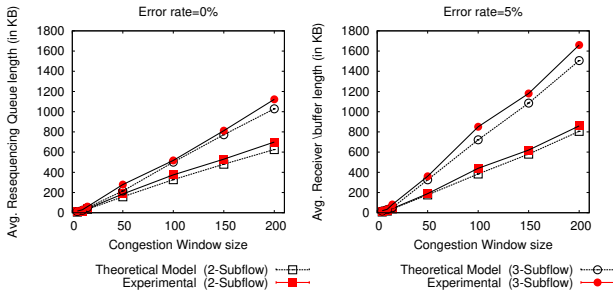


Fig. 4: Receiver Buffer Size Comparison

Subsequently, we can approximate the average receiver buffer length ($E_{RL}(\vec{Q})$) for a given configuration $\vec{Q}$ based on [19]:

$$E_{RL}(\vec{Q}) \approx (\max_k(r_k) - \min_k(r_k)) Avg_{Th}(\vec{Q}) \tag{17}$$

E. Model Verification

To verify the correctness of our proposed DTMC based model, we have compared the average throughput and receiver buffer length with emulation results obtained using Mininet [20]. The test topology is given in Fig. 2. All the switches given in the topology (Sw1-Sw6) are SDN switches. The emulation links are configured to have $20ms$ delay. Path S1 and S2 have bottle neck bandwidth of $8mbps$ and S3 is configured with $18mbps$ of bandwidth. Results are obtained for two different loss rates (0% and 5%). Fig. 3 shows the effect of maximum congestion window size with average throughput for two and three active sub-flows. Fig. 4 represents the effect of maximum congestion window size on the length of the receiver buffer. We observe that our proposed model can predict the behavior of MPTCP with significant confidence. Therefore, in the next section, we present the sub-flow scheduling problem based on this estimation model.

IV. Sub-Flow Selection based on Performance Estimation from DTMC

The objective of sub-flow selection problem, for a given MPTCP connection and a set of all available sub-flows $\mathcal{S} = \{S_1, S_2 \dots S_n\}$, is to select a subset of $\mathcal{S}$ for optimizing the average throughput. However, optimal average throughput can increase the receiver buffer size, which in turn might deteriorate the overall performance. Therefore, sub-flow selection problem must limit receiver buffer size to a certain threshold (RL_{max}). The length of the receiver buffer length depends upon the congestion control algorithm and scheduling mechanism. Therefore, in the previous section, we propose a mathematical model to estimate receiver buffer length in Eq. (17) in presence of BALIA congestion control. The proposed model also provides the average throughput (Eq. (15)). Based on the estimated values, the sub-flow selection problem can be formulated as a mixed integer linear program (MILP).

Given $\mathcal{S}$, the power set of $\mathcal{S}$ ($\wp(\mathcal{S})$) provides all the possible configurations. Let, $\vec{I}$ be the indicator vector of all possible sub-flow configuration such that $\vec{I} = \{\forall k \in \wp(\mathcal{S} : I^k)\}$. We define $I^k \in \mathbb{R}^n$ for a given configuration $k \in \wp(\mathcal{S})$ as per Eq. (18).

$$I_j^k = \begin{cases} 1 & \text{if } S_j \in k \\ 0 & \text{otherwise} \end{cases} \tag{18}$$

Let, $\vec{Q} = \{q_1, q_2, ..., q_n\}$ be the path quality matrix of all available sub-flows such that $q_i = \{b_i, l_i, \mu_i, \sigma_i\}$. Therefore, the effective path quality matrix of only active sub-flows (X^k) for the k^{th} configuration can be expressed as $X^k = \vec{Q} \circ I^k$ where $\circ$ denotes the Hadamard product (element wise product) between two matrices of same dimension. We denote the average throughput of all active sub-flows in k^{th} configuration as $Avg_{Th}(X^k)$. $Avg_{Th}(X^k)$ can be calculated using Eq. (15). Eq. (17) can be used to calculate the estimated receiver buffer length ($RL(X^k)$) for the k-th configuration. Now we can represent the optimal sub-flow selection problem as an optimization problem as given in Eq. (19).

$$\begin{aligned} &\max_k Avg_{Th}(X^k) \\ &\text{subjected to, } RL(X^k) \leq RL_{max} \end{aligned} \tag{19}$$

This optimization problem is equivalent to 0-1 knapsack problem [21], where $Avg_{Th}(\vec{X})$ and $RL(\vec{X})$ can be treated as the capacity of the knapsack. 0-1 knapsack problem is known to be *NP*-hard. Therefore, we propose a greedy heuristic Alg. 1 to solve Eq. (19).

We define the effective bandwidth of a sub-flow as $b_i(1-l_i)$. Our proposed heuristic should be able to increase the effective bandwidth. However, as per Eq. (17), the length of the receiver buffer inversely proportional to the effective bandwidth. Eq. (17) also reveals that, with the increase in RTT, delay between key segment and the rest of the segment increases. Therefore, we can conclude that RTT of a sub-flow is directly proportional to the length of receiver buffer length. So, the proposed heuristic is built upon these two governing factors. We apply linear scalarization to find the best possible sub-flow. Our proposed heuristic ensures that a sub-flow with high effective bandwidth and low RTT gets higher priority of selection if that sub-flow does not increase estimated receive buffer length than RL_{max}.

Input: $\vec{Q}$
Output: $\vec{I}$
$\forall i : I_i \leftarrow 0$;
Sort $\vec{Q}$ based on $T_i \leftarrow b_i(1 - l_i) + \frac{1}{\mu_i}$;
Find $\max_i(T_i)$; $I_i \leftarrow 1$;
for $j \leftarrow 2$ ***to*** n **do**
  $\vec{X} \leftarrow \vec{Q} \circ I$; $\mathcal{A} \leftarrow Avg_{Th}(\vec{X})$; $\mathcal{R} \leftarrow RL(\vec{X})$
  if $\mathcal{R} \leq RL_{max}$ **then**
    $I_j \leftarrow 1$;
  end
end
return $\vec{I}$;

Algorithm 1: Heuristic for sub-flow selection

To implement the heuristic, we exploit SDN capabilities for accumulating $\vec{Q}$. In case of SDN supported infrastructure, an SDN controller may periodically gather individual port statistics such as link bandwidth, loss rate and approximate delay for each data plane device. The gathered statistics can provide an estimate of end-to-end characteristics. Upon receiving a MPTCP connection-open request, the controller finds the set of n paths between the source and the destination based on the underlying routing protocol. The value of n depends on the number of network interfaces available and the path manager used by the end hosts. According to the full-mesh path manager, all of the n paths should be used as active sub-flows. After the initial path selection and sub-flow identification, the controller periodically calculates the end-to-end quality of sub-flow S_i (as denoted by Q_i). Upon calculating $\vec{Q}$, the controller uses Alg. 1 to calculate the set of active sub-flows as $\mathcal{S}_{active} = \{\forall i, I_i \neq 0 : S_i\}$. This $\mathcal{S}_{active}$ is relayed back to the path manager which activates the corresponding sub-flows.

V. Implementation Details and Performance Evaluation

In this section, we discuss the performance of the proposed sub-flow selection mechanism in previous section. We have emulated an SDN environment through *Open vSwitch* [22] via the Mininet [20] emulation platform at the Department of Computer Science and Engineering, IIT Guwahati.

A. Implementation Methodology

For our emulation environment, we use POX controller [23] to implement the heuristic proposed in Alg. 1. To manage switch statistics, we modify the POX "`flow_stat`" module. The statistics are stored in Tinydb [24] database. Once the controller detects a topology change event, the controller invokes sub-flow selection module. The module recalculates the active sub-flow set for the affected flows and pro-actively notifies the source path manager framework via UDP in a `JSON` format. In case a new flow enters the system, the controller uses "`L3_learning`" routing protocol to find the available paths for the newly generated flow. Once the sub-flow establishment is done, sub-flow selection module is invoked

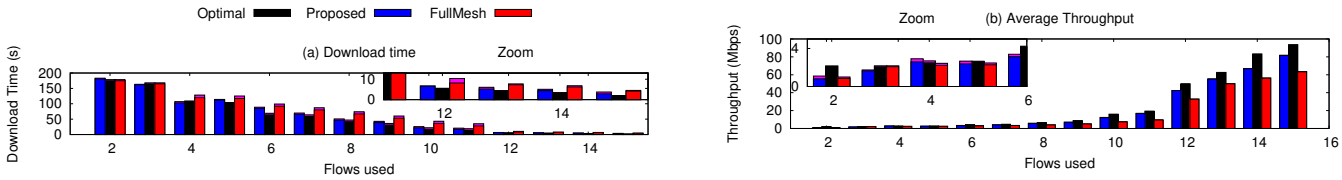


Fig. 5: Emulation Results – Flow Completion Time and Throughput

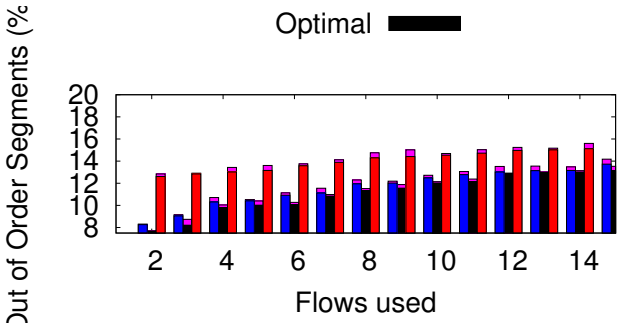


Fig. 6: Out of Order Segments

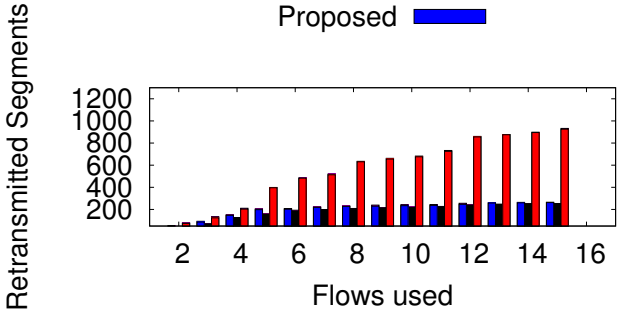


Fig. 7: Retransmitted Segments

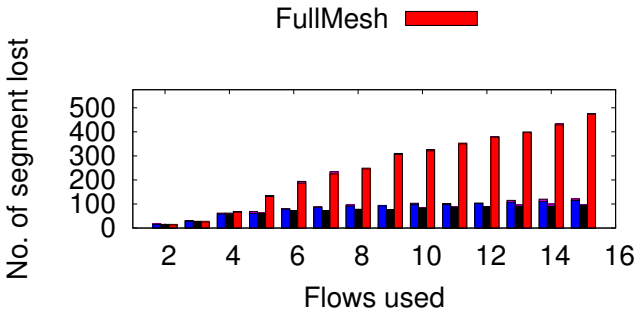


Fig. 8: Lost Segments

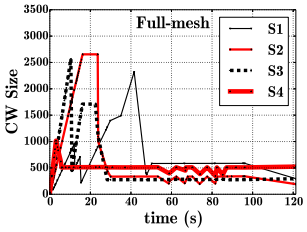


Fig. 9: Congestion window size

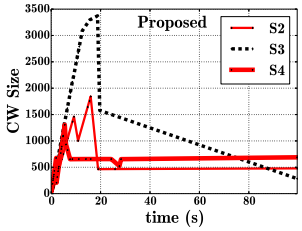


Fig. 10: Contention Window Variation

and the active sub-flow set is recalculated for the flows which have at least one common link with the newly generated flow.

We have used open-source MPTCP kernel module [2] in our testbed. To integrate with the SDN POX controller, our developed module[1] uses UDP to communicate. Upon detecting change, POX recalculates the active sub-flow set and pushes to the host as a recommendation in JSON format. The recommendation identifies the sub-flow set by the network addresses. Upon receiving the request, the path manager module translates network addresses to sub-flow IDs and selects the corresponding sub-flows as active. Accordingly, it notifies the congestion control module about these changes.

B. Competing Heuristics

It can be noted that to the best of our knowledge, existing literature have not worked on the MPTCP sub-flow selection problem. As discussed earlier, the MPTCP kernel implementation has two variants of sub-flow selection or path manager algorithm – *Full-mesh* and *ndiffports*. Although *ndiffports* progresses in the direction of sub-flow selection, but it uses a naive implementation of random sub-flow selection, which does not work well in practice. Therefore, we consider the *Full-mesh* path manager as the competing heuristic of our proposed protocol. Further, we compare the performance of the proposed methodology with the optimal performance, as computed by enumerating over all possible combination of paths.

C. Topology Details and Emulation Results

Topology – We choose a topology which is similar to the one given in Fig. 2. We configure 15 parallel paths between a sender and a receiver with the end-to-end parameters as follows. We increase the bandwidth of these paths from 1 Mbps to 15 Mbps with a step of 1 Mbps. The delay is increased from 10 ms to 150 ms with a step of 10 ms, whereas the path loss increases from 0% to 15% with a step of 1% per path. The sender generates MPTCP supported HTTP flows destined towards receiver host.

Average file download time and aggregated throughput – Fig. 5 shows the performance comparison of the proposed scheme with the Full-mesh path manager and off-line optimal path manager in terms of download time of a $100MB$ file over standard HTTP protocol and the average aggregated throughput. The emulation results reveal that, although Full-mesh path manager performs quite well for up to 3 sub-flows, further increase in the number of sub-flows increases the download

[1]https://github.com/subhrendu-subho/SDN_pathmanager

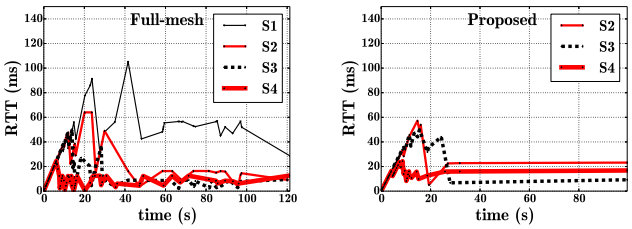


Fig. 11: RTT Variation

time significantly and reduces the average throughput. The performance of the proposed methodology is considerably well compared to the Full-mesh path manager, and also very close to the optimal performance as we observe for our emulation scenarios.

Effect on congestion control parameters – To understand why the proposed methodology significantly boosts up the performance of MPTCP, we explore the evolution of several parameters that control MPTCP congestion control mechanism. As given in Fig. 6, a significant reduction in out of order segments can be observed in case of our proposed methodology in comparison to the Full-mesh path manager. As shown in Figs. 7 and 8, our proposed path manager also significantly reduces the number of retransmitted segments and lost segments by selecting effective sub-flows.

Analysis of congestion window evolution – The reason behind the effectiveness of the proposed sub-flow management mechanism can be justified by the help of the congestion window progression analysis. Fig. 10 shows the progression of the congestion window for Full-mesh path manager when it uses 4 sub-flows. For the similar scenario, the proposed path manager uses only 3 sub-flows which can be observed in Fig. 10. As argued earlier, due to the reduction of lower bandwidth path, the proposed methodology can reduce the number of retransmit events along with the out of order segments. Therefore, it can help all the sub-flows to converge to their steady state congestion window size quickly.

Impact on RTT – On the other hand, our proposed scheme also reduces the receiver buffer size. Therefore, the sub-flows experience lesser delay compared to the Full-mesh path manager, and observe reduced RTT, as shown in Fig. 11. As a result, the overall performance improves significantly with the help of the proposed sub-flow management module hooked with the standard MPTCP kernel.

VI. Conclusion

In this work, we develop a sub-flow management framework for MPTCP protocol over a SDN controlled enterprise-grade or data-center networks. Our proposed framework reduces out of order segments and HOL blocking in MPTCP. The emulation results show that our proposed sub-flow management heuristic outperforms the existing path manager in MPTCP and very closely approximates a *NP*-hard problem of optimal sub-flow selection in terms of various performance metrics.

References

[1] A. Ford, C. Raiciu, M. Handley, S. Barre, and J. Iyengar, "Architectural guidelines for multipath tcp development," IETF, RFC6824, Tech. Rep., Jan 2011.

[2] "MultiPath TCP - Linux Kernel implementation : Main - Home Page browse," Jul 2018, [Online; accessed 16. Jul. 2018]. [Online]. Available: https://multipath-tcp.org

[3] C. Raiciu, M. Handley, and D. Wischik, "Coupled congestion control for multipath transport protocols," IETF, RFC6356, Tech. Rep., 2011.

[4] R. Khalili, N. Gast, M. Popovic, and J.-Y. Le Boudec, "MPTCP is not pareto-optimal: performance issues and a possible solution," *IEEE/ACM Tran. on Networking*, vol. 21, no. 5, pp. 1651–1665, 2013.

[5] A. Walid, Q. Peng, S. H. Low, and J. Hwang, "Balanced Linked Adaptation Congestion Control Algorithm for MPTCP," Internet Engineering Task Force - Draft, Tech. Rep., Jan 2016.

[6] C. Xu, J. Zhao, and G. M. Muntean, "Congestion Control Design for Multipath Transport Protocols: A Survey," *IEEE Comm. Surveys & Tutorials*, vol. 18, no. 4, pp. 2948–2969, Fourthquarter 2016.

[7] D. Wischik, C. Raiciu, A. Greenhalgh, and M. Handley, "Design, implementation and evaluation of congestion control for multipath tcp," in *Proceedings of the 8th USENIX Conference on Networked Systems Design and Implementation*, ser. NSDI'11. Berkeley, CA, USA: USENIX Association, 2011, pp. 99–112.

[8] M. Li, A. Lukyanenko, S. Tarkoma, Y. Cui, and A. Yl-Jski, "Tolerating path heterogeneity in multipath tcp with bounded receive buffers," *Computer Networks*, vol. 64, pp. 1 – 14, 2014.

[9] K. Xue, J. Han, H. Zhang, K. Chen, and P. Hong, "Migrating unfairness among subflows in mptcp with network coding for wired–wireless networks," *IEEE Transactions on Vehicular Technology*, vol. 66, no. 1, pp. 798–809, 2017.

[10] Y. E. Guo, A. Nikravesh, Z. M. Mao, F. Qian, and S. Sen, "Accelerating multipath transport through balanced subflow completion," in *Proc. of the 23rd ACM MobiCom*, 2017, pp. 141–153.

[11] Y.-s. Lim, E. M. Nahum, D. Towsley, and R. J. Gibbens, "Ecf: An mptcp path scheduler to manage heterogeneous paths," in *Proc. of the 13th ACM CoNEXT*, 2017, pp. 147–159.

[12] Y. Cao, Q. Liu, Y. Zuo, F. Ke, H. Wang, and M. Huang, "Receiver-centric buffer blocking-aware multipath data distribution in MPTCP-based heterogeneous wireless networks." *KSII Tran. on Internet & Information Systems*, vol. 10, no. 10, 2016.

[13] S. Chattopadhyay, S. Nandi, S. Shailendra, and S. Chakraborty, "Primary path effect in multi-path tcp: How serious is it for deployment consideration?" in *Proc. of the 18th ACM MobiHoc*, 2017, pp. 36:1–36:2.

[14] J. Kim, B. H. Oh, and J. Lee, "Receive buffer based path management for mptcp in heterogeneous networks," in *2017 IFIP/IEEE IM*, May 2017, pp. 648–651.

[15] Y. Zhang, H. Mekky, Z.-L. Zhang, F. Hao, S. Mukherjee, and T. Lakshman, "Sampo: Online subflow association for multipath tcp with partial flow records," in *Proc. of 35th IEEE INFOCOM*, 2016, pp. 1–9.

[16] D. Kreutz, F. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-Defined Networking: A Comprehensive Survey," *Proc. of the IEEE*, vol. 103, no. 1, pp. 14–76, Jan 2015.

[17] B. Nunes, M. Mendonca, X. Nguyen, K. Obraczka, and T. Turletti, "A survey of software-defined networking: Past, present, and future of programmable networks," *IEEE Communications Surveys Tutorials*, 2014.

[18] Q. Peng, A. Walid, J. Hwang, and S. H. Low, "Multipath TCP: Analysis, design, and implementation," *IEEE/ACM Tran. on Networking*, vol. 24, no. 1, pp. 596–609, 2016.

[19] F. Wang, D. Xie, J. Wang, P. Zhang, and Y. Shi, "Paths selection-based resequencing queue length in concurrent multipath transfer," *Int. Journal of Communication Systems*, vol. 28, no. 11, pp. 1805–1827, 2015.

[20] B. Lantz, B. Heller, N. Handigol, V. Jeyakumar, and B. OConnor, "Mininet-an instant virtual network on your laptop (or other pc)," 2015.

[21] H. M. Salkin and C. A. De Kluyver, "The knapsack problem: A survey," *Naval Research Logistics Quarterly*, vol. 22, no. 1, pp. 127–144, 1975.

[22] "Open vSwitch," May 2018, [Online; accessed 15. Jul. 2018]. [Online]. Available: http://www.openvswitch.org

[23] "noxrepo/pox," Jul 2018, [Online; accessed 16. Jul. 2018]. [Online]. Available: https://github.com/noxrepo/pox

[24] "Introduction TinyDB 3.9.0.post1 documentation," Jul 2018, [Online; accessed 16. Jul. 2018]. [Online]. Available: http://tinydb.readthedocs.io/en/latest/intro.html

PTC: Pick-Test-Choose to Place Containerized Micro-services in IoT

Shubha Brata Nath*, Subhrendu Chattopadhyay†, Raja Karmakar‡,
Sourav Kanti Addya§, Sandip Chakraborty¶ and Soumya K Ghosh‖
*§¶‖Indian Institute of Technology Kharagpur, India † Indian Institute of Technology Guwahati, India
‡Techno International New Town, India
Email: *nath.shubha@gmail.com, †subhrendu@iitg.ac.in, ‡rkarmakar.tict@gmail.com,
§kanti.sourav@gmail.com, ¶sandipc@cse.iitkgp.ac.in, ‖skg@cse.iitkgp.ac.in

***Abstract*—In the presence of the Internet of Things (IoT) devices, the end-users require a response within a short amount of time which the cloud computing alone cannot provide. Fog computing plays an important role in the presence of IoT devices in order to meet such delay requirements. Though beneficial in these latency-sensitive scenarios, the fog has several implementation challenges. In order to solve the problem of micro-service placement in the fog devices, we propose a framework with the objective of achieving low response time. This problem has been formulated as an optimization problem to improve the response time by considering the time-varying resource availability of the fog devices as constraints. We propose an orchestration framework named *Pick-Test-Choose* (PTC) to solve the problem. PTC uses *Bayesian Optimization based iterative reinforcement learning* algorithm to find out a micro-service allocation based on the current workload of the fog devices. PTC employs containers for service isolation and migration of the micro-services. The proposed architecture is implemented over an in-house testbed as well as in iFogSim simulator. The experimental results show that the proposed framework performs better in terms of response time compared to various other baselines.**

***Keywords*-Fog Computing; Internet of Things; Micro-service Placement; Container Placement; Reinforcement Learning**

I. Introduction

Usage of cloud infrastructure is a popular choice for the deployment of large scale distributed applications. However, the cloud is not a suitable platform for many Internet of Things (IoT) applications. Short bursty flows generated by the IoT applications increase the response time when each of the packets is processed in the cloud. To overcome this limitation, Cisco proposed *"Fog computing"* [1], which employs *"in-network processing"* to deliver user desired quality of service (QoS) for time critical IoT applications. Fog hosts small-scale cloud to support *"in-network processing"* by using the residual resources of the network equipment like the routers. Fog devices are resource constrained; therefore, instead of using monolithic *"service oriented architecture"* (SOA), the fog applications or services must adopt *"micro-service architecture"* [2], where each application is developed in the form of a bunch of loosely coupled lightweight services. Although the placement of micro-services over various fog devices is as important as the service placement in case of cloud computing, it is not very straightforward and differs from the traditional service offloading mechanisms [3], [4]. The primary challenges are as follows. (i) As the fog uses in-network processing, all the fog devices have their primary workloads (e.g., routing for a network router), and the fog resource availability varies over time. Further, the secondary workload from the micro-services should not interfere with the primary workload of the fog devices. (ii) Multiple micro-services can be placed simultaneously at the fog devices; therefore, the architecture requires micro-service isolation to ensure resource provisioning. (iii) Based on the temporal nature of the primary workload, the architecture must support seamless migration of micro-services from one fog device to another to ensure application QoS. (iv) The deployment framework and the micro-service migration framework must be lightweight so that it can cater to low latency IoT applications by deciding micro-service placement quickly. (v) The highly dynamic nature of the in-network processing architecture increases monitoring overhead, and maintaining consistency of the monitored statistics is difficult. The monitoring inconsistency leads to the performance degradation of the overall system. Ensuring these five requirements simultaneously is non-trivial over a fog environment.

A few works in the recent literature have addressed the problem of micro-service placement in the fog devices. In [5], the authors have addressed the application placement problem by placing the virtual machines (VM) in the edge or fog devices. However, the use of VM requires high resource consumption; to avoid this overhead, [6] have proposed a container (lightweight virtualization technology) driven framework to speed-up application deployment procedure. [7] and [8] have proposed the container placement and module mapping algorithm respectively in a hierarchical fog environment. DROPLET [9] provides a computation partitioning and offloading procedure. On the other hand, a micro-service offloading framework by exploiting traditional memory allocation strategies have been proposed in [10]. In Foglets [11], the authors have proposed a greedy service placement strategy for a hierarchical fog infrastructure in the presence of mobile users to minimize the access delay. For this type of mobility capable fog systems, Mobility-based [12] have provided a VM placement and migration framework to maximize the number of applications placed in fog while reducing the overall application latency. However, most of these existing works

978-1-7281-0962-6/19/$31.00 ©2019 IEEE

are primarily targeted towards a dedicated fog infrastructure where the fog device does not have any primary workload. So, the existing works are incapable of handling the temporal primary workload demands. Therefore, the design of a QoS ensuring micro-service deployment framework with resource constraints is non-trivial in case of fog deployment.

In this paper, we develop a lightweight orchestration framework named Pick-Test-Choose (PTC) for dynamic online micro-service placement and resource management over a fog computing infrastructure. PTC (§II) solves the above five requirements over a dynamic time-constrained environment by using containers [13]. Use of container ensures – (i) service isolation and (ii) lightweight migration of micro-services from one fog device to another as required. To solve the dynamic placement problem of micro-services over fog devices based on resource availability and workload characteristics, we formulate the placement problem as an optimization problem based on integer linear programming (ILP) (§III). However, ILP is NP-hard even with a static placement, and therefore, it is difficult to find an optimal placement when the temporal workload pattern is difficult to estimate. We consider this time-varying effect of the primary workload is the inherent environmental factor and can be learned by using *"environmental learning"* [14] primitive.

For this purpose, we use a reinforcement learning based algorithm where the framework can monitor and predict the nature of resource availability and workload characteristics over time (environmental factors), and it can dynamically decide on selecting a suitable fog device for executing a micro-service. The proposed algorithm can also readjust the placement based on the change in the environmental factors. Consequently, we use Bayesian Optimization [15], a lightweight, dynamic, iterative, and online reinforcement learning framework as a design choice for solving the micro-service deployment problem (§III-E). The proposed framework has been implemented and tested over a fog networking prototype testbed. We observe that the proposed framework can reduce the average application response time than the baselines. Also, the proposed framework provides better functionalities in terms of micro-service isolation and lightweight execution. To test the scalability of the framework, we have implemented the proposed framework over *iFogSim* [16] simulator. The experimental results have shown that our framework is effective for optimization of system response time meeting the resource constraints.

II. System Architecture

The system architecture, as shown in Figure 1, has the following components.

Client Node: Client nodes are end-user nodes. The task of a client node includes the generation of a fog service request.

Fog Controller Device: The fog controller device is responsible for managing the computation offloading in the fog devices. A fog controller is logically connected with the fog devices. Fog controller identifies the client requests and maps the request to a suitable application through `application mapper`. This application is further divided into micro-services by the `micro-service organizer` module, following the existing approaches such as ENTICE [17]. The organizer module is also responsible for the containerization of the micro-services to overcome the micro-service isolation issue. Once the micro-services are identified, it is the task of the `micro-service dispatcher` to identify the fog devices which can host the micro-services. To identify a feasible micro-service placement, the dispatcher module takes help of the `statistics collector module`. The major task of the statistics collector is to periodically monitor the available resources of the connected fog devices. The proposed framework is capable of finding out the placement of micro-services even in the presence of noises in the measurement. Once the placement schedule is generated by the dispatcher module, the micro-services are delegated to the fog devices by a `micro-service migration engine`. During the execution of the micro-services, the fog devices might get overloaded by its primary workloads. In such cases, the micro-service dispatcher and the migration engine handle the situations by the regeneration of placement schedule and migrating micro-services according to the schedule.

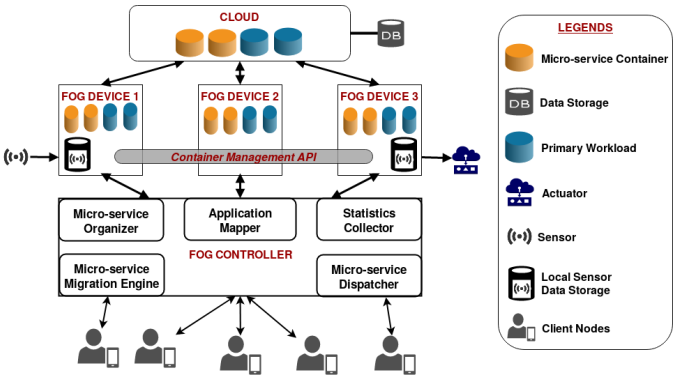


Fig. 1. In-network processing architecture with PTC framework

Fog Device: Fog devices are responsible for the actual computation and storage related to the applications provided by the fog-cloud. Fog devices use container technologies for resource isolation and management of the micro-services executing in them. General purpose fog devices are mainly equipment like networking hardware, smart meters, gateways, etc. Therefore, the fog devices have some primary workloads which cannot be affected at any cost. These primary workloads consume resources based on environmental demands. In Figure 1, we represent these primary workloads with blue colored containers. The residual capacity of the fog devices can be utilized by the fog micro-service containers, delegated by the controller management modules. Orange colored containers in the Figure 1 represent such containers. The micro-services running in the containers might require interaction with sensors and actuators. Our framework contains a dedicated `local sensor data storage` module to ease this process. During the execution of the micro-services, a container may require migration from one fog device to

another to satisfy resource demand of the primary workload. To support the migration, we use a distributed `container management` application programming interface (API).

Cloud: Although the primary objective of fog is to place micro-services as near as possible to the data sources to reduce communication overhead, sometimes it is not possible to cater to all the micro-service demands inside the fog. Therefore, the proposed framework keeps the provision of using the cloud to avoid such overloaded scenarios.

III. Micro-service Placement as an Optimization

We consider an undirected and weighted communication graph $G = (V, E, W)$. Here, V represents the set of physical nodes, E is the set of physical communication links, and W is the set of edge weights. We denote $S = (S_i \in (1, \ldots, s))$, $\Lambda = (\Lambda_i \in (1, \ldots, \lambda))$, and $F = (F_i \in (1, \ldots, n))$ as the sets of sensors, actuators, and fog devices respectively. Here, s, λ, and n are the total number of sensors, actuators, and fog devices respectively. Therefore, $V = \{S \cup \Lambda \cup F\}$. $e_{i,j} \in E$ represents the physical communication link between $v_i \in V$ and $v_j \in V$. In this case, the weight of each edge signifies delay of the links. $D(i1, i2)$ defines the weight of the shortest path between v_{i1} and v_{i2}, which signifies the communication delay between the nodes. We consider that the time is divided into l time slots where each slot is of length ι. Therefore, we define system time vector $T = (\tau_t : t \in (1, \ldots, l))$.

Let us consider that there are k applications present in the system such that $\vec{\Psi} = (P_x : x \in (1, \ldots, k))$. $\vec{\Psi}$ is the set of applications. We consider that an application P_x can further be divided into h_x atomic micro-services, and each micro-service can execute independently. We define $p_{x,y}$ as the y^{th} micro-service of P_x. Hence, $P_x = (p_{x,y} : y \in (1, \ldots, h_x))$. We denote the resource vector available in F_i as $\vec{R}(F_i)$, and $\vec{\Gamma}(p_{x,y})$ denotes the resource vector required by micro-service $p_{x,y}$. We assume that the resource vector required by $p_{x,y}$ does not change over time. $\vec{\Gamma}(p_{x,y}) = (\gamma^q_{x,y} \in \mathbb{R} : q \in (1, \ldots, f))$, where each component $\gamma^q_{x,y}$ signifies the amount of resource type (central processing unit, memory, network bandwidth, etc.) q required to execute $p_{x,y}$, and f is the total type of resources. Similarly, $\vec{R}(F_i) = (r^q_i \in \mathbb{R} : q \in (1, \ldots, f))$, where r^q_i is the total available quantity of resource type q at F_i. Each fog device F_i can host several micro-services based on the available resources. A micro-service allocation matrix $[A] = (A_{x,y,i,t} \in (0,1) : (x \in (1, \ldots, k), y \in (1, \ldots, h_{max}), i \in (1, \ldots, n), t \in (1, \ldots, l))$ provides the mapping between micro-services and fog devices at each time quantum, where $h_{max} = \max_x(h_x)$, and each element $A_{x,y,i,t} = 1$ if $p_{x,y}$ is assigned to F_i at time t, otherwise $A_{x,y,i,t} = 0$. Let $O(A, x, y)$ signify the occupancy of fog device resources by $p_{x,y}$ at each time instance as defined in Eq. (1).

$$[O(A,x,y)]_{n\times l} = (A_{x,y,i,t} \in (0,1) : i \in (1 \ldots n), t \in (1 \ldots l)) \tag{1}$$

The total response time for an application has the following components.

A. Estimation of Processing Time

Each micro-service cannot occupy more than one fog device at the same time. Therefore, the row vector $Seq_{exe}(x,y) = [I]_{1\times n} \times [O(A,x,y)]_{n\times l}$ provides the fog device execution sequence by $p_{x,y}$, where $[I]_{1\times n} = [1, 2, \ldots, n]$ represents row vector of the fog device indices. If F_i is capable of executing δ_i million instructions per second (MIPS), and $p_{x,y}$ requires $\xi_{x,y}$ million instructions, the processing time (T_{CPU}) required by $p_{x,y}$ can be expressed as follows.

$$T_{CPU}(x,y,A) = \sum_{i \in Seq_{exe}(x,y)} \frac{\xi_{x,y}}{\delta_i \iota} \tag{2}$$

B. Estimation of Migration Time

Let $M_{x,y} = (M^b_{x,y} = Seq_{exe}(x,y)_b : Seq_{exe}(x,y)_b \neq Seq_{exe}(x,y)_{b+1})$ represent the sub-vector of $Seq_{exe}(x,y)$ where b is a particular time slot. $M_{x,y}$ represents the fog device migration vector by $p_{x,y}$. Therefore, the total time required for migration (T_{MGR}) is

$$T_{MGR}(x,y,A) = \sum_{M^b_{x,y}} (\Delta_{x,y} \times D(M^b_{x,y}, M^{(b+1)}_{x,y})) \tag{3}$$

Here, $\Delta_{x,y}$ denotes a constant delay factor for migration of $p_{x,y}$, which depends upon the amount of data to be communicated from one fog device to another, and $D(v_i, v_j)$ represents the weight of the shortest path between v_i and v_j.

C. Estimation of Data Fetch and Actuation Time

Let $\psi = (\psi_{x,y,i} : x \in (1, \ldots k), y \in (1, \ldots h_{max}), i \in (1, \ldots s))$ and $\alpha = (\alpha_{x,i} : x \in (1, \ldots k), i \in (1, \ldots \lambda))$ represent the demand vector of sensors and actuators respectively by the micro-services. Here, $\psi_{x,y,i} = 1$ if $p_{x,y}$ requires S_i, otherwise 0. Similarly, $\alpha_{x,i} = 1$ if P_x requires Λ_i and $\alpha_{x,i} = 0$ for all other cases. We denote sensors and actuators required by $p_{x,y}$ as $\psi_{x,y} = [I]_{1\times s} \times (\psi_{x,y,i} : i \in (1, \ldots s))$ and $\alpha_x = [I]_{1\times\lambda} \times (\alpha_{x,i} : i \in (1, \ldots \lambda))$ respectively. $[I]_{1\times s} = [1, 2, \ldots, s]$ represents the row vector of the sensor indices, and $[I]_{1\times\lambda} = [1, 2, \ldots, \lambda]$ represents the row vector of the actuator indices. We define the data fetch time (T_{DF}) of $p_{x,y}$ as follows.

$$T_{DF}(x,y,A) = \max_{i \in \psi_{x,y}} (D(M^1_{x,y}, i)) \tag{4}$$

We define the time required to send a reply to actuators (T_{ACT}) as follows.

$$T_{ACT}(x,y,A) = \max_{i \in \alpha_x} (D(M^{|M_{x,y}|}_{x,y}, i)) \tag{5}$$

Let $\Omega(A, i, t)$ signify the executing micro-service vector at fog device F_i at time t, and it can be defined as follows.

$$\Omega(A,x,y,i,t) = (A_{x,y,i,t} \in (0,1) : x \in (1 \ldots k), y \in (1 \ldots h_{max})) \tag{6}$$

From Eq. (6), at time t, we can calculate the amount of occupied resource type q as follows.

$$\Theta(A, i, q, \vec{\Gamma}, t) = \left(\sum_{x,y} \left(\Omega(A,x,y,i,t) \times \gamma^q_{x,y} \right) \right) \tag{7}$$

According to the capacity property, cumulative occupied resources by micro-services executing on a single fog device must not surpass the total available resource at that fog device. A valid allocation matrix must satisfy the capacity property as given in Eq. (8).

$$\underset{i,q,t}{\forall} \Theta(A, i, q, \vec{\Gamma}, t) \leq r_i^q \tag{8}$$

Therefore, the total response time required by $\vec{\Psi}_x$ can be calculated by using Eqs. (2) to (5), as follows.

$$T_{Resp}(A, x) = \max_{y \in (1 \ldots h_{max})} \begin{pmatrix} T_{DF}(x,y,A) + T_{CPU}(x,y,A) \\ + T_{MGR}(x,y,A) + T_{ACT}(x,y,A) \end{pmatrix} \tag{9}$$

D. Problem Definition

Given the communication graph G and available resources ($\vec{R}(F_i)$), the micro-service placement problem finds an allocation schedule (A) for each micro-service ($p_{x,y}$) with required instructions ($\xi_{x,y}$) and resources ($\vec{\Gamma}(p_{x,y})$) such that the maximum response time taken by the applications is minimized. Therefore, we have

$$\begin{aligned} &\underset{A}{\text{minimize}} \qquad T_{Resp}^{max}(A) \\ &\text{subject to:} \\ &\qquad \mathcal{R}(A) \geq \vec{Z} \end{aligned} \tag{10}$$

where $T_{Resp}^{max}(A) = \max_x (T_{Resp}(A, x))$, $\mathcal{R}(A)_{i,q,t} = \vec{R}(F_i) - \Theta(A, i, q, \vec{\Gamma}, t)_{i,q,t}$, and $\vec{Z} = (z_{i,q,t} = 0 : i = (0, \ldots, n), q = (0, \ldots, f), t = (0, \ldots, l))$.

Based on the formal definition of the problem, we can prove that the *"Minimax Facility location problem"* (MFLP) [18] is polynomial time reducible to micro-service allocation problem given in Eq. (10). Thus, the micro-service allocation is $\mathcal{NP}$-hard. However, we exclude this proof due to the space constraints of this paper.

E. Solution Approach: Micro-service Placement using Bayesian Optimization

As the problem is $\mathcal{NP}$-hard and difficult to implement due to high monitoring overhead, we propose a reinforcement learning framework which requires very little monitoring and can perform in the presence of noise. For this purpose, we design *"Bayesian optimization"* (BO) based mechanism, PTC, to place micro-services over the fog devices. BO optimizes the objective function based on the prior observations and posterior distribution by conducting iterative experiments over the solution configurations. In our case, conducting an experiment is equivalent to test the performance of a given allocation matrix which is costly in terms of time taken to perform a test. BO performs well in such cases where performing one single experiment takes higher time.

Without loss of generality, we assume that the utility function $T_{Resp}^{max}(A)$ follows a normal distribution. Based on this prior belief function, BO executes initial experiments. After sufficient amount of preliminary experiments, BO updates the prior belief function based on the posterior distributions. To update the prior distribution, BO uses *"acquisition function"* (EI) which intelligently notifies BO to choose the configurations for subsequent experiments. This leads the framework towards optimum configuration. Let $\mathcal{D}_d = \{(a_1, T_{Resp}^{max}(a_1)), \ldots, (a_d, T_{Resp}^{max}(a_d))\}$ be the set of prior observations after d iterations. We denote $p(T_{Resp}^{max}) = \mathcal{N}(\mu, K)$. Here, the mean function is $\mu(\circ)$, and the co-variance kernel function is $K(\circ, \circ)$. These are defined as follows.

$$\mu(a_u) = \mathbb{E}(T_{Resp}^{max}(a_u)) \tag{11}$$

$$K(a_u, a_v) = \mathbb{E}\left(\left(T_{Resp}^{max}(a_u) - \mu(a_u)\right)\left(T_{Resp}^{max}(a_v) - \mu(a_v)\right)\right) \tag{12}$$

Let us denote Φ and ϕ as the standard normal cumulative distribution function and the standard normal density function respectively. We define $U_{min} = \min_{a \in \mathcal{D}_d}(T_{Resp}^{max}(a))$, $\pi = \frac{U_{min} - \mu(a_d)}{\sigma(a_{d-1}, a_d)}$, and $\sigma(a_{d-1}, a_d) = \sqrt{K(a_{d-1}, a_d)}$. We choose our acquisition function as described in Eq. (13) as recommended in [19].

$$EI(A|\mathcal{D}_d) = \begin{cases} 0 & \text{if:} \sigma(a_{d-1}, a_d) = 0 \\ ((U_{min} - \mu(a_d))\,\Phi(\pi)) + (\phi(\pi)\sigma(a_{d-1}, a_d)) & \\ & \text{Otherwise} \end{cases} \tag{13}$$

However, $EI(A|\mathcal{D}_d)$ works well in case of unconstrained optimization. Therefore, to satisfy our constrained optimization, we adopt the procedure suggested by Gardner et al. [20]. By following their footsteps, we assume that $\mathcal{R}(\mathcal{A})$ follows Bernoulli process, and EI can be modified to Eq. (14).

$$EI^c(A|\mathcal{D}_d) = P\left(\mathcal{R}(A)\right) EI(A|\mathcal{D}_d) \tag{14}$$

In our set-up, there can be observation noise due to various factors such as a rise in loads in the fog devices, network delays due to channel states, the rise in reporting delays due to various circumstance, etc. Therefore, the proposed BO algorithm must withstand observation noise. It is a standard practice to assume observation noise (ζ) as a normally distributed random variable with zero mean i.e., $\zeta = \mathcal{N}(0, \sigma_\zeta)$.

IV. Evaluation

The proposed framework is implemented both in a testbed and over a simulation environment. The testbed gives results under a realistic setup, whereas the simulation has helped us to understand the scalability of PTC orchestration framework.

A. Implementation Details

The testbed is implemented using `Raspberry Pi 3 model b` (https://www.raspberrypi.org/products/raspberry-pi-3-model-b/) single-board computers as the fog devices, connected over a local network. Raspberry Pi 3 model b is equipped with quad core 1.2 GHz Broadcom BCM2837 64 bit central processing unit (CPU) and 1 GB random access memory (RAM). Therefore, these devices have low processing and storage capability. Our deployed fog

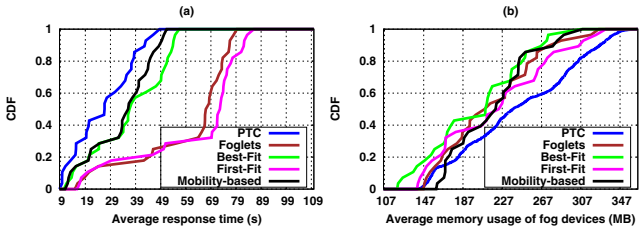


Fig. 2. (a) CDF of average response time and (b) CDF of average memory usage of the fog devices

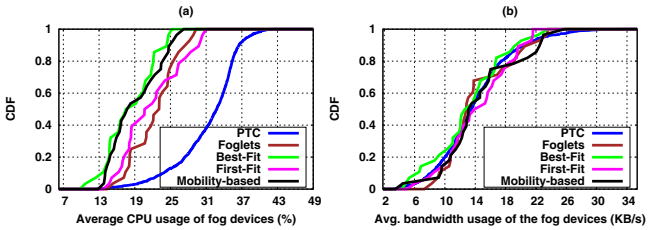


Fig. 3. (a) CDF of average CPU usage of the fog devices and (b) CDF of average network bandwidth usage of the fog devices

devices use `Ubuntu 16.04.2 long-term support (LTS)` (https://www.ubuntu.com/) operating system. In the fog devices, we use `Docker` (https://www.docker.com/) to achieve micro-service isolation. Docker provides a platform for lightweight containers by application layer virtualization. The framework is also connected with a private cloud environment available in our institute.

B. Results

Figure 2(a) illustrates the performance of PTC in terms of average response time of the applications. We observe that the average response time is significantly low in PTC compared to other schemes; therefore, the cumulative distribution function (CDF) of PTC converges within 50 seconds. On the other hand, CDFs of other competing mechanisms converge between 53 seconds and 88 seconds. Through deep dive into the performance logs, we observe that BO in PTC converges quickly to an optimal or near-optimal allocation vector (within 30 iterations). BO has a tendency to find the desired points in the search space with relatively few function evaluations. With the help of this optimization technique, adaptive learning is induced in PTC, and thus, PTC becomes an intelligent scheme. In this regard, BO-based adaptive learning mechanism helps PTC to optimize the utility function with the help of prior observations and posterior distribution such that the overall response time can be reduced. As a result, the framework learns the fog environment and can take action accordingly to provide a minimum average response time for running the applications.

The CDF of the average memory usage of the fog devices is shown in Figure 2(b). The PTC has more average memory usage than the baseline schemes. The density of such distribution is higher (in the range of 139 MB – 366 MB) in PTC than that of the baselines. Whereas, the baselines converge within 334 MB for first-fit, within 327 MB for foglets, within 310 MB for mobility-based, and within 308 MB for best-fit respectively. The CDF of average CPU usage of the fog devices is shown in Figure 3(a). It is higher in PTC, and it is in the range of 12% – 44%. The baselines converge within 27% for best-fit, within 33% for first-fit, within 31% for foglets, and within 29% for mobility-based algorithm respectively. The CDF of bandwidth usage of the fog devices is shown in Figure 3(b). It is higher in PTC. It is in the range of 3 KB/s – 36 KB/s. Whereas, such distributions converge within 25 KB/s for the best-fit and foglets mechanisms. It converges within 23 KB/s for the first-fit mechanism and within 27 KB/s for the mobility-based algorithm.

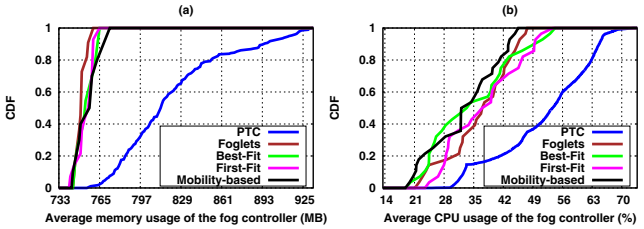


Fig. 4. (a) CDF of average memory usage of the fog controller device and (b) CDF of average CPU usage of the fog controller device

Next, we look into the average resource usage by the fog controller which is responsible for running the BO based placement algorithm. The CDF of the average memory usage by the fog controller is shown in Figure 4(a). PTC has more average memory usage for the controller device due to the overhead of multiple iterations. The density of average memory usage distribution of controller is higher (in the range of 756 MB – 933 MB) in PTC than that of the baselines. The average memory usage distribution of the fog controller converges within 760 MB for foglets, within 765 MB for best-fit, within 766 MB for first-fit, and within 774 MB for mobility-based algorithm. The CDF of average CPU usage of the controller device is shown in Figure 4(b). It is higher in PTC, and it is in the range of 26% – 74%. The baselines converge within 56% for best-fit, within 55% for first-fit, within 49% for foglets, and within 47% for mobility-based algorithm respectively. The CDF of the average bandwidth distribution for PTC controller is shown in Figure 5(a). It is higher (in the range of 11 KB/s – 291 KB/s). The baselines converge within 74 KB/s for best-fit, within 68 KB/s for first-fit, within 69 KB/s for foglets, and within 69 KB/s for the mobility-based mechanism.

PTC uses container migration for balancing the load across the fog devices with an objective towards minimizing the average response time of the application. We have shown the average number of migrations of the micro-services in Figure 5(b). PTC migrates the micro-services in a new fog device only when there is a better fog device available in terms of resource availability. Interestingly, we observe that though the total number of migrations increases, the average number

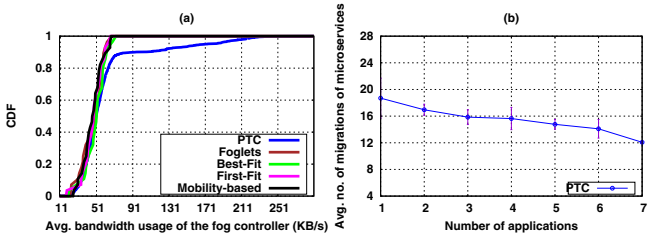


Fig. 5. (a) CDF of average network bandwidth usage of the fog controller device and (b) Avg. number of migrations of the micro-services

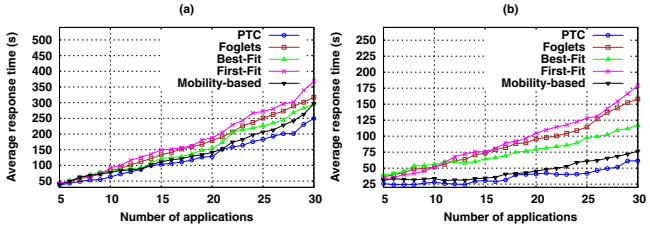


Fig. 6. Average response time: (a) number of fog devices=20, and (b) number of fog devices=40

of migrations decreases with the increase in the number of applications.

We have also implemented PTC in iFogSim [16] in order to check the scalability of the proposed framework. The simulation is performed with an increased numbers of fog devices.

Figure 6 shows the average response times under different number of fog devices. In order to optimally place the micro-services in fog devices, we need an online learning which can select the available fog devices dynamically by considering the load distribution as well as the resource availability of the devices. In addition, an optimization technique is required to place the micro-services in fog devices in real-time and migrate these services in proper fog devices such that the average response time of the applications gets reduced. In this regard, BO is an iterative reinforcement learning approach which tries to optimize its utility function based on the knowledge gained through the execution of the algorithm. This adaptive learning based optimization helps PTC to utilize the available fog devices as much as possible, such that the maximum number of applications can be placed parallelly. As a result, the average response time is reduced significantly in PTC, as the number of available fog devices increases.

V. Conclusion

In this paper, we have proposed a BO-based iterative reinforcement learning mechanism for containerized micro-service placement in IoT considering the time-varying resource availability based on the fog device workloads. Based on the primary workload of the fog devices, there is a need to select the correct set of devices for micro-service placement in order to minimize the response time of the applications. In order to achieve this requirement, we have formulated the micro-service assignment problem as an optimization problem which is solved using a Bayesian Optimization based iterative reinforcement learning algorithm. We have implemented PTC in an in-house testbed setup as well as in *iFogSim* simulator, and it is observed that PTC can minimize the response time of the system. In the future, we plan to evaluate the performance of PTC under different applications.

References

[1] F. Bonomi, R. Milito, P. Natarajan, and J. Zhu, "Fog computing: A platform for internet of things and analytics," in *Big data and internet of things: A roadmap for smart environments*. Springer, 2014, pp. 169–186.

[2] S. Hassan and R. Bahsoon, "Microservices and their design trade-offs: A self-adaptive roadmap," in *Proc. of the 13th IEEE SCC*. IEEE, 2016, pp. 813–818.

[3] G. Orsini, D. Bade, and W. Lamersdorf, "Computing at the mobile edge: Designing elastic android applications for computation offloading," in *Proc. of the 8th IFIP WMNC*. IFIP, 2015, pp. 112–119.

[4] M. Aazam, S. Zeadally, and K. A. Harras, "Offloading in fog computing for IoT: review, enabling technologies, and research opportunities," *Future Generation Computer Systems*, vol. 87, pp. 278–289, 2018.

[5] W. Wang, Y. Zhao, M. Tornatore, A. Gupta, J. Zhang, and B. Mukherjee, "Virtual machine placement and workload assignment for mobile edge computing," in *Proc. of the 6th IEEE CloudNet*. IEEE, 2017, pp. 1–6.

[6] A. Ahmed and G. Pierre, "Docker Container Deployment in Fog Computing Infrastructures," in *Proc. of the IEEE EDGE*. IEEE, July 2018, pp. 1–8.

[7] E. Yigitoglu, M. Mohamed, L. Liu, and H. Ludwig, "Foggy: A Framework for Continuous Automated IoT Application Deployment in Fog Computing," in *Proc. of the 6th IEEE AIMS*. IEEE, 2017, pp. 38–45.

[8] M. Taneja and A. Davy, "Resource aware placement of IoT application modules in Fog-Cloud Computing Paradigm," in *Proc. of the 15th IFIP/IEEE IM*. IFIP/IEEE, 2017, pp. 1222–1228.

[9] T. Elgamal, A. Sandur, P. Nguyen, K. Nahrstedt, and G. Agha, "DROPLET: Distributed Operator Placement for IoT Applications Spanning Edge and Cloud Resources," in *Proc. of the 11th IEEE CLOUD*. IEEE, July 2018, pp. 1–8.

[10] V. Souza, X. Masip-Bruin, E. Marín-Tordera, S. Sànchez-López, J. Garcia, G.-J. Ren, A. Jukan, and A. J. Ferrer, "Towards a proper service placement in combined Fog-to-Cloud (F2C) architectures," *Future Generation Computer Systems*, vol. 87, pp. 1–15, 2018.

[11] E. Saurez, K. Hong, D. Lillethun, U. Ramachandran, and B. Ottenwälder, "Incremental deployment and migration of geo-distributed situation awareness applications in the fog," in *Proc. of the 10th ACM DEBS*. ACM, 2016, pp. 258–269.

[12] D. Gonçalves, K. Velasquez, M. Curado, L. Bittencourt, and E. Madeira, "Proactive virtual machine migration in fog environments," in *Proc. of the 23rd IEEE ISCC*. IEEE, 2018, pp. 00 742–00 745.

[13] D. Bernstein, "Containers and cloud: From lxc to docker to kubernetes," *IEEE Cloud Computing*, vol. 1, no. 3, pp. 81–84, 2014.

[14] S. Gough and W. Scott, *Sustainable Development and Learning: Framing the Issues*. Routledge, 2003.

[15] J. Snoek, H. Larochelle, and R. P. Adams, "Practical bayesian optimization of machine learning algorithms," in *Advances in neural information processing systems*, 2012, pp. 2951–2959.

[16] H. Gupta, A. Vahid Dastjerdi, S. K. Ghosh, and R. Buyya, "iFogSim: A toolkit for modeling and simulation of resource management techniques in the Internet of Things, Edge and Fog computing environments," *Weily Software: Practice and Experience*, vol. 47, no. 9, pp. 1275–1296, 2017.

[17] G. Kecskemeti, A. C. Marosi, and A. Kertesz, "The ENTICE approach to decompose monolithic services into microservices," in *Proc. of the 14th IEEE HPCS*. IEEE, 2016, pp. 591–596.

[18] Z. Drezner and G. O. Wesolowsky, "Minimax and maximin facility location problems on a sphere," *Naval Research Logistics Quarterly*, vol. 30, no. 2, pp. 305–312, 1983.

[19] O. Alipourfard, H. H. Liu, J. Chen, S. Venkataraman, M. Yu, and M. Zhang, "Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics," in *Proc. of the 14th USENIX NSDI*. USENIX, 2017, pp. 469–482.

[20] J. R. Gardner, M. J. Kusner, Z. E. Xu, K. Q. Weinberger, and J. P. Cunningham, "Bayesian Optimization with Inequality Constraints," in *Proc. of the 31st ICML*, 2014, pp. 937–945.

Fast and Secure Handoffs for V2I Communication in Smart City Wi-Fi Deployment

Pranav Kumar Singh(✉), Subhredu Chattopadhyay, Pradeepkumar Bhale, and Sukumar Nandi

Department of Computer Science and Engineering, Indian Institute of Technology, Guwahati 781039, India
{pranav.singh,subhrendu,pradeepkumar,sukumar}@iitg.ernet.in

Abstract. The Intelligent Transport System (ITS) is a vital part of smart city developments. Due to densely deployed access points and vehicular mobility in a smart city, the number of handovers also increases proportionately. Minimization of the handoff latency is crucial to provide a better quality of service for vehicles to have access different ITS services and applications. Increased handover latency can cause an interruption in vehicle-to-infrastructure (V2I) communication. In this paper, we propose a fast and secure handoff mechanism for smart cities that have acceptable handoff latency for delay-sensitive ITS applications and services. Our proposal considers mobility and communication overhead to provide lower handoff latency. We compare our proposed mobility aware background scanning mechanism (AdBack) with standard Active Scanning mechanism in an emulated test bed. Our test results reveal that the proposed AdBack mechanism significantly outperforms the existing mechanisms in terms of handover latency, packet drop rates, and throughput. Experimental results show that amalgamation of AdBack and existing fast re-authentication (IEEE 802.11r) can improve connectivity for V2I communication in a smart city. We provide rigorous emulation results to justify the performance of our proposed scheme.

1 Introduction

Vehicular communication that emerged from wireless communication has gained much interest from academia, industries, and governments to improve road safety, fuel efficiency, and convenience of travel. Vehicular communication is one of the leading research areas because of its applications and its specific characteristics. In the wake of the Information and communication technology (ICT) revolutions, road transportation has entered into a new era, and today we have technologies and services such as connected vehicles, driverless cars, smart cars, VANET, Internet of vehicles, vehicle telematics, and intelligent transportation systems (ITS). Based on the data collected from various studies, surveys, polls and driver's experiences hundreds of applications can be suggested. Most of

A. Negi et al. (Eds.): ICDCIT 2018, LNCS 10722, pp. 189–204, 2018.
https://doi.org/10.1007/978-3-319-72344-0_15

these vehicular applications fall into three main categories: safety, non-safety, and infotainment.

- **Safety Applications:** The road safety can be provided by vehicle-to-vehicle (V2V), vehicle-to-pedestrian (V2P) and vehicle-to-infrastructure (V2I) communication. However, safety-critical applications such as post-crash warning, pedestrian crossing, lane change warning, emergency brake, and do-not-pass warning are provided by V2V and V2P mode.
- **Non-Safety Applications:** Vehicles with a set of sensors can provide a wide variety of sensed information related to the vehicle, traffic, environment, parking and driving conditions. Collected sensor information can be uploaded from vehicles to a smart city road administration database using the roadside infrastructures (access points) in V2I mode. Vehicles traveling on the same route can obtain information for their applications.
- **Infotainment Applications:** Vehicle-to-Network (V2N) communications enable infotainment applications such as Voice over Internet Protocol (VoIP), video download, streaming, live TV, software update of the in-vehicle unit, map update, instant messaging, cloud-based services and Internet access. We consider V2I for this category as well. These applications can improve driving comfort and keep vehicle occupants informed. The maximum allowable latency for the VoIP application can be up to 50 ms. The allowed latency and the range of communication required for these application categories can be found in [1].

With the increasing number of vehicles and development of the smart cities, new issues related to traffic jam, road accidents, and carbon emission have emerged. Vehicular communication can help solve these problems and bring innovations in these contexts. Many countries worldwide, have already deployed early stages of vehicular networks to make transport safer and comfortable.

With urbanization and the development of smart cities, there is rapid growth in Wi-Fi deployments that make Wi-Fi a complementary, low-cost solution for V2I/I2V connectivity. Providing Quality of Service (QoS), seamless connectivity and security in densely deployed scenarios of a smart city Wi-Fi are some of the biggest challenges. The communications range of the road-side access points is a maximum of 300–400 m. The handover delay at Layer 2 due to the scanning and reauthentication delay can disrupt ongoing communication (such as VoIP) of vehicle occupants in a smart city. The Wi-Fi protocol stack (IEEE 802.11a/b/g/n/ac) is not designed for the vehicular mobility context. However, various amendments to the IEEE 802.11 standard is bringing advancements in Wi-Fi. Such as, IEEE 802.11r provides fast re-authentication [2], IEEE 802.11i [3] for enhanced security, IEEE 802.11w provides protection to management frames, IEEE 802.11e for QoS, etc. These amendments does not solve the vehicular mobility problem. The wireless network architecture must provide fast, secure, seamless, and highly available connectivity to its users, regardless of whether they are static or moving. In this paper, we demonstrate that our proposed mobility aware scanning with IEEE 802.11r (rather than full IEEE 802.11i scanning) can provide seamless connectivity to the V2I services.

The rest of the paper is organized as follows: Sect. 2 describes the background to our topic. We analyze the work related to Level 2 handover latency minimization in Sect. 3. We present our proposal for scanning mechanism in Sect. 4. Section 5 covers the details of the emulation setup used to demonstrate our proposed mechanism. In Sect. 6, we analyze the performance in a Wi-Fi deployment of a smart city in terms of handover latency, packet loss, and average throughput. Finally, we conclude our work in Sect. 7.

2 Background

This section presents a summary based on comprehensive analysis of IEEE 802.11 management frames that are responsible for maintaining communication between access points (APs) and wireless clients during the static as well as handover procedure. We present the detailed background of handover management which includes discovery and reauthentication in wireless network protected with Wi-Fi Protected Access II (WPA2: 802.1X/Extensible Authentication Protocol (EAP)) mechanism.

2.1 Discovery Mechanism

Handoff due to vehicular mobility is the process of reassociation to the new AP when a vehicle moves away from the currently associated AP, and the received signal strength decreases to a certain threshold. The discovery process of the new AP involves initiation and scanning. As the mobile node moves away from the connected AP, the Received Signal Strength Indicator (RSSI) begin to drop and force the mobile node to discover new accessible APs. Of all scanned APs, the mobile node selects one for its association based on some specific criteria. There are two types of the mechanism that allows the mobile node to discover target AP: Passive Scanning and Active Scanning.

1. **Passive Scanning:** In passive scanning mode, the mobile nodes listen for the beacon management frames broadcasted by the APs. Beacon frames are transmitted periodically by the AP to announce its presence. The default broadcasting interval is usually configured as 100 ms and is known as the Beacon Interval. Thus, it may take 100 ms for a mobile node to hear a beacon frame. Passive scanning usually takes more time, since the mobile node has to wait long enough on a channel for a beacon frame. Because it is a time-consuming process to hear a beacon frame, most mobile nodes prefer an active scan.
2. **Active Scanning:** In active scanning mode, the mobile node switches to a new channel and broadcasts Probe Request frames on it and waits for the Probe Response frames from APs operating on that channel. If no response received on that channel, it is assumed empty, and the mobile node switches to a new channel. This process repeats for all operating channels. The set of the channel depends on mode and country. Finally, received Probe Response frames are processed by the mobile node to obtain information about candidate access point.

The research studies [4,5] have already measured the scanning delay, which suggests that it varies between 600 ms to 700 ms. They observed that discovery delay is the dominating component of the handoff delay. It accounts more than 90% of the overall handoff delay. Thus probing is the bottleneck for fast handoff and should be reduced to provide seamless connectivity.

In case of an wireless infrastructure based WPA2 Enterprise network, every handover mechanism must be followed by a reauthentication procedure after the scanning. The reauthentication phase that includes key management is equally time-consuming. It varies between few milliseconds to second [4], depends on which authentication mode (Pre Shared Key (PSK) or 802.1X/EAP) and protocol used. Authors [6] in their performance study have shown that if 802.1X/EAP authentication (baseline 802.11i authentication) is used then the average roaming time is 525 ms and maximum consecutive lost datagrams (Average) is 53.

The handover due to mobility can severely affect QoS and QoE for real time applications and ITS services in the 802.11i Enterprise based security framework. Thus, to minimize latency during reauthentication and key management the IEEE Task Group r (TGr) was formed.

2.2 IEEE 802.11r Fast BSS Transition

The 802.11r standard amendment specifies a Fast Basic Service Set Transition (FT-BSS) mechanism ratified in 2008.

This section describes the IEEE 802.11r security framework and FT-BSS transition process.

Fast BSS Transition Security Framework. The handover process based on the 802.1X/EAP security framework consists of 6 phases: initiation, discovery, 802.11 open authentications, reassociation, reauthentication, and the key-handshake. The (re)authentication phase of WPA2 Enterprise based on 802.1X/EAP uses an external server (e.g., Remote Authentication Dial-In User Service (RADIUS)) to provide Authentication, Authorization, and Accounting (AAA). Without FT enabled, the mobile node needs to go through a complete reauthentication (including key management) after reassociation in each handover. In the FT framework of IEEE 802.11r, reauthentication is performed efficiently before reassociation.

As per the specification of the current draft of IEEE 802.11r, 802.1X/EAP based authentication is done once when the mobile node initially joins the network and generates the Pairwise Master Key (PMK). The generated PMK is distributed to all APs belonging to the same mobility domain. Thus, this presence of PMK at all APs helps to reduce reauthentication delay that incurs in communication to an external server (RADIUS) for authentication.

For a mobile node that is 802.11r compatible, the 4-way handshake followed by the QoS request over WLAN using IEEE 802.11e is completed during the reassociation phase, further reducing the overall handover latency. In contrast, an 802.11i-based mobile node needs to repeat full 802.1X authentication and

4-way handshake during every handover. If QoS is enabled, then there will be more frame exchange in IEEE 802.11i, which will contribute to an additional delay to the overall latency.

FT BSS Transition. FT BSS transition is the process of disassociating from one and re-associating to new AP, and all the APs belong to the same mobility domain (same Extended Service Set (ESS)). The set of frame exchanges in reauthentication and key-handshake takes a considerable amount of time in a secure WLAN based on 802.1X/EAP. Thus, the number of the frame exchange between a mobile node and an AP must be reduced during the transition. It will help minimizing interruption to delay-sensitive services such as voice and video during the handover from one AP to another. There are two underlying FT protocols used for subsequent re-associations to APs within the same mobility domain. These two are described as follows:

FT Protocol: FT protocol is for a simple transition of the mobile node that does not require resource request before its transition.

FT Resource Request Protocol: In this protocol, the mobile node requires a resource request before its transition. In this paper, we consider FT Protocol only (without resource request) in this work. There are two methods of Fast BSS transition: Over-the-Air and Over-the-Distribution System (DS) Fast BSS Transition. A mobile node can opt one of these for it's handover to a target AP (selected after scanning) from the currently associated AP.

Over-the-Air Fast BSS Transition: In this Fast BSS Transition, mobile node directly communicates with the target AP over the air. Only four frames are exchanged between the mobile node and the target AP during reauthentication. They contain appropriate information for PTK generation at the both the end. Now, time-critical phases 802.1X/EAP including 4-way key-handshake are not required to unblock the uncontrolled port.

Over-the-DS Fast BSS Transition: In Fast BSS Transition, mobile node communicates with the target AP via the current AP. Communication between a mobile node and the target AP takes place using FT Action frames.

Over-the-Air vs Over-the-DS: In case of Over-the-Air (OTA), the mobile node needs to leave its active channel to negotiate on another channel during scanning. The mobile node sends a frame to its currently associated AP and tells it to go into sleep mode. When the negotiation completes, then it returns to the active channel to flush its and AP's buffer. The OTA can interrupt communication in a place where the mobile node is already at the edge of the AP range, suffering from poor performance.

In the Over-the-DS mode, the mobile node does not leave the channel. The mobile node stays on its current channel and asks the current AP to negotiate with the next AP. However, mobile node still needs to discover the target AP first by using some scanning mechanism. Over-the-DS mechanism improves performance in terms of lower BSS transition time than the OTA mechanism.

3 Related Work

The early research work, [5,7], tried to solve the problem related to the high probe delay observed in [4] using selective scanning, caching and neighbor graph. However, these mechanisms not tested in the context of vehicular mobility. The method proposed in [7] requires changes inside the presently deployed 802.11 APs. Authors of [8] have focused on the same problem using interleaved scanning in a random mobility. The public HotSpots region is selected covered by several APs with more than 20% of the overlapping area.

Studies in [9,10] proposed multiple wireless cards for AP and the mobile device, respectively. The mechanisms proposed in these studies are not practical to the same technology access and could be expensive as well.

The research works in [11–14] target to reduce Layer-2 handoff latency by adopting synchronization and pre-scan mechanism. However, these works used the passive approach, and the complexity of implementation is quite high.

The researchers of [15–17] have used a prediction of node mobility to improve performance and provide a better connectivity. The prediction mechanisms requires information such as position and movement direction, geolocation, and mobility history, respectively. Since determining correct position of the vehicle is not that easy, forecasting a better connectivity in a highly dynamic vehicular environment accurately is a difficult task. The navigation driven algorithms proposed in [18] may not be suitable for the vehicular context because vehicles have to move on the defined road topology and cannot change their route immediately depending on the handoff decision. The handoff strategy used in this work is a lazy type, where the handoff initiation occurs only when a mobile node disconnected with currently associated AP.

Finally, in [19–22] work is done related to the vehicular context. The researchers of [19] have used a directional antenna and beam steering techniques to collect information on a particular route, which in practice not feasible. The handover protocol proposed in [20] is complex in its implementation and [21] again used a prediction mechanism based on historical information.

In [22], the authors have analyzed the security properties and performance of IEEE 802.11i, IEEE 802.11r, HandOver KEY (HOKEY) and Control and Provision of Wireless Access Points (CAPWAP) for handoff in V2I communication. Studies in [6,23,24] tested and analyzed Layer-2 handover delay due to re-authentication only. Authors in [25] have compared the performance of IEEE 802.11r with Legacy IEEE 802.11 but the details of the discovery phase based on the location mechanism are not provided.

Most of the research work on Layer-2 handover scheme contributes only on minimization of the discovery delay (search or finding target AP) and does not consider the re-authentication delay part. In this paper, we are proposing a simple and fast scanning mechanism and test it with IEEE 802.11r-2008 as well.

4 Proposed Mechanism

From the discussion in the earlier section, primary contributing factors of handover latency are discovery delay and re-authenticating delay. In this paper, we provide a novel scheme for minimizing discovery delay of target roadside unit (RSU). Notations used in our proposed algorithm are listed in Table 1.

Table 1. Notations used in Algorithm 1

Symbol	Description
$RSSI_{RSU}$	RSSI of the RSU
$RSSI_{th}$	RSSI Threshold specified in bgscan modes
T_s	Short-interval for scanning
T_l	Long-interval for scanning
$Channels_{RSU}$	Database for scanned AP information in Learn Mode
STA_{speed}	Current speed of the Vehicle
$Speed_{th}$	Vehicle's maximum speed
BGScanLearn()	Learn Mode of the bgscan
BGScanSimple()	Simple Mode of the bgscan

4.1 Adaptive Background Scanning

As mentioned earlier (in Sect. 3) that the scanning phase is the bottleneck for fast handoff. Thus, ProbeDelay has to be reduced to provide seamless handover.

Handover strategies for vehicular communication need to be mobility aware. The reason to consider the mobility is that it severely affects performance in wireless networks. Therefore, we propose our Adaptive Background Scanning scheme (AdBack) to support fast and seamless roaming in densely deployed APs. The proposed AdBack scheme is mobility aware that relies on bgscan [26]. The On-board Unit (OBU) of vehicles usually equipped with a set of sensors including Gyro sensor, a processing unit, memory, and storage. Today, even our smartphones come with a set of sensors like Proximity, Gyro, light, accelerometer, digital compass, and magnetometer as well. Sensors can provide three crucial information: movement, direction, and speed. We are using speed information to improve connectivity and overall performance. Our mechanism is Adaptive because it adapts to different speeds which is detected by sensor. In Proposed AdBack algorithm, we are using movement information, which in a real scenario, can be provided by the accelerometer sensor. The handover decision can be improved by the use of accelerometer sensor data.

Periodic Background Scanning. In background scanning (bgscan), the mobile node scans channels to roam within an ESS (i.e., within a single network block). Other criteria of this mechanism are that all the APs in the ESS

should have same Service Set Identifier (SSID). The bgscan provides three different modes. In None mode, the background scanning is disabled. The Simple mode enables periodic background scanning based on $RSSI_{th}$. When $RSSI_{RSU}$ is greater than or equals to the $RSSI_{th}$ perform background scanning after every T_l and when the $RSSI_{RSU}$ is less the $RSSI_{th}$ perform scanning after the T_s. In Learn mode of bgscan, the mobile node learns channels used by the network and try to avoid bgscans on other channels which reduces the effect on the data connection. A mobile node in Learn mode maintains a $Channels_{RSU}$.

We describe our proposed AdBack scanning scheme in Algorithm 1. AdBack relies on STA_{speed} in addition to $RSSI_{th}$ for the handover decision. When the vehicle is static, proposed AdBack does not perform scanning unless it reaches to $RSSI_{th}$, as it might interrupt some of the ongoing communication unnecessarily. If AdBack detects vehicle is moving at slow speed, it switches to the Simple mode and performs periodic background scanning. We assign fixed values to T_s and T_l, which can be derived from simple calculation on STA_{speed} and communication range of the RSU. Finally, if the vehicle is moving at high speed, daemon switches to Learn mode, where it tries to associate RSU learned previously (maintained as $Channels_{RSU}$) and avoids any interruption in communication due to scanning. The running daemon on OBU switches to Simple mode only if the vehicle is not able to associate RSUs present in $Channels_{RSU}$.

We claim that the proposed AdBack scheme provides better performance in terms of handover latency, packet loss and average throughput. We provide an emulation experiment to ascertain our claim.

5 Experimental Setup

To justify our claim for our proposed fast and secure handoff mechanism, we use Mininet-WiFi emulator [28]. This section covers implementation in Mininet-WiFi, selected reference scenario and parameters used for performance analysis in our experiment.

5.1 Implementation in Mininet-WiFi

We implemented our proposal AdBack scanning mechanism and existing standard IEEE 802.11r in Mininet-WiFi. A detailed description related to our implementation given at Mininet-WiFi discussion forum [29]. We installed freeradius server on Ubuntu and integrated with Mininet-WiFi. We implemented IEEE 802.11r and AdBack scanning (modified bgscan) in userspace that makes it more flexible and enables faster implementation than kernel-space. The association control is implemented for proper execution of handover due to mobility. The traffic simulator Simulation of Urban Mobility (SUMO) [27] is used to model mobility. For handover latency and packet loss analysis, the vehicle speed is fixed to 14 m/s. For the average throughput analysis, we have assigned random speed to the vehicles, which includes stoppage and slowdown. The varying mobility

```
Function BGScanSimple(RSSI_RSU, RSSI_th, T_s, T_l):
    while true do
        if RSSI_RSU ≥ RSSI_th then
            Wait for T_l;
            Scan;
        else
            Wait for T_s;
            Scan;
        end
    end
return
Function BGScanLearn(RSSI_RSU, RSSI_th, T_s, T_l):
    BGScanSimple(RSSI_RSU, RSSI_th, ∞, ∞);
    Channels_RSU ← Store Channels with active RSUs;
    while RSSI_RSU ≤ RSSI_th do
        Scan channels ch|ch ∈ Channels_RSU;
        if RSU Available then
            Associate with RSU;
        else
            BGScanSimple();
        end
    end
return
Input: RSSI_RSU, RSSI_th, T_s, T_l, Speed_th
BGScanSimple();
if STA_speed == 0m/s then
    if RSSI_RSU ≤ RSSI_th then
        BGScanSimple();
    else
        Do not scan;
    end
else
    if 0 < STA_speed < Speed_th then
        BGScanSimple();
    else
        BGScanLearn();
    end
end
```

Algorithm 1. Proposed AdBack Scheme

helps us to model parking, braking and stoppage at the traffic light, fuel station and service center. We created the network topology for selected reference scenario.

5.2 Reference Scenario

As depicted in Fig. 1, we have taken smart city Wi-Fi setup as our reference scenario. Our scenario consists of set of RSUs deployed alongside the road. All RSUs are connected through a DS and belong to the same ESS. In the ESS, the vehicle performs intra-domain handover when it moves across RSUs. We exported Pune city road segment from an open street map (OSM) to model real traffic scenario using SUMO. The Corresponding Node represents the server installed at smart-city road authority to maintain real-time traffic information. The vehicle driver tries to fetch that data from the server while it is moving across those RSUs in the given road segment. An AAA server is maintained by the smart citys road administration to allow an authorized vehicle to use applications and services in a secured manner.

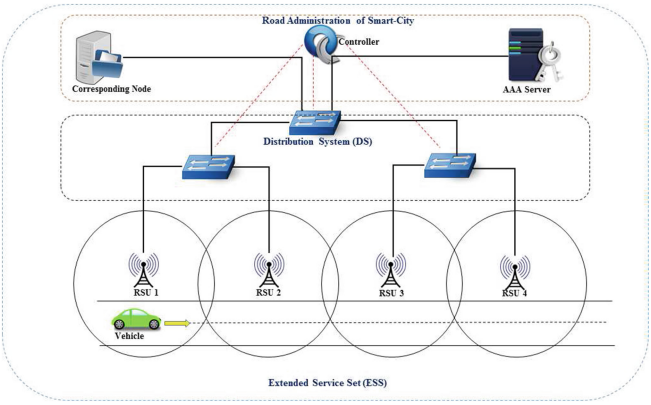


Fig. 1. Reference scenario of a smart city

5.3 Simulation Parameters

We run our simulation for 200 s on a smart city road segment of length approximately 3000 m. For realistic urban modeling, we have used Log Distance propagation loss model. We use SUMO for traffic modeling with varying speed, and maximum limit is set 50 Kmph (approx. 14 m/s). We are using 802.11g that operates in 2.4 GHz band and provide speed up to 54 Mbps. The RSUs have an overlapping coverage area of 20% of its radio range. Overlapping RSUs operate in non-overlapping channels. We have a set of 3 non-overlapping channels, i.e., a combination of 1, 6 and 11. Ten RSUs cover our target region. The vehicle is initially associated with RSU1 in a smart city network; the vehicle and the corresponding node communicates over IP in WLAN. Vehicular mobility ($Speed_{th}$) and signal strength threshold ($RSSI_{th}$) triggers handover when it moves across different RSUs deployed within the same ESS.

All our simulation parameters and modeling are close to the realistic scenario and as per the smart city Wi-Fi requirements. Table 2 shows details of simulation parameters used in our experiment.

Table 2. Simulation parameters

Parameters	Values
Operating system	Ubuntu 14.04-LTS
AAA server	FreeRADIUS Version 2.1.12
Traffic simulator	SUMO
Wi-Fi emulator	Mininet-WiFi
Wired link parameters	Bandwidth: 100 Mbps, Propagation Delay: 5 ms
RSU antenna type	Omnidirectional
Propagation model	Log Distance Propagation Loss Model
Path loss exponent	3.5 (Urban)
Simulation area	Approx 3 Km
Number of RSUs	10
Maximum velocity	14 m/s
Radio range of RSU	300 m
Wireless mode	IEEE 802.11g, Data rate: 54 Mpbs, RTS/CTS enabled
Authentication mode	WPA-Enterprise: 802.1X/EAP and FT-EAP
Authentication protocol	EAP-TLS
Traffic type	VoIP
Protocols used	ICMP, TCP
WLAN security framework tested	IEEE 802.11i, IEEE 802.11r FT over-DS
Scanning mechanism	Active, and AdBack
Simulation duration	200 s
Performance metrics	Packet Loss, Handoff delay and Avg. Throughput

6 Performance Evaluation

In WLAN based V2I communication, the QoS metrics can be Layer-2 handoff latency, packet loss, and throughput. The analysis of these parameters is helpful to evaluate the performance of V2I communication in a smart-city Wi-Fi deployment. In this section, the performance of our mechanism as well as the Legacy approach is assessed on Mininet-WiFi and compared based on these metrics.

6.1 Handoff Latency and Packet Loss with VoIP Like Traffic

The handoff latency is the time when the handover decision was made by the vehicle to join new RSU, and when successfully associated with the new RSU. In our experiment, VoIP like packets transmitted and received from the mobile node to the corresponding node. Packets are Internet Control Message Protocol (ICMP) packets since we are creating similar traffic using the ping utility. Packets of size 80 bytes are generated at every 20 ms to model VoIP like traffic. The objective is to identify handover latency and packet loss for VoIP communication during the handover process. In our experiment, nine handoffs performed, and we recorded the results of handover latency and packet loss. During a passage from one RSU to another, a mobile node continuously communicates with the corresponding node on the network using ICMP packets generated via ping utility at the rate of one per 20 ms.

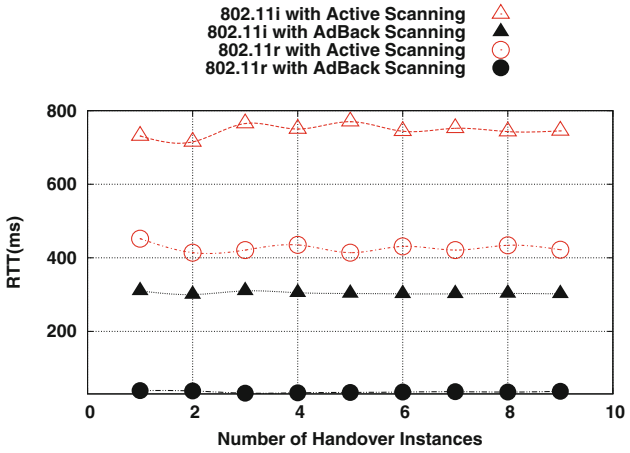


Fig. 2. Handover latency in milliseconds

In Fig. 2, we provide the simulation results for AdBack scanning and compare it with standard Active Scanning mechanism when used with IEEE 802.11i and IEEE 802.11r. We use round trip time (RTT) in milliseconds (ms) to measure the handover latency of the combinations mentioned above. We can see that the AdBack scanning with the IEEE 802.11r security framework is one of the fastest that takes approx. 35 ms in handover while other combinations take more than 300 ms. The proposed combination of scanning and reauthentication reduces the handover latency to the extent required for delay-sensitive applications such as VoIP.

Figure 3, shows that the AdBack scanning with IEEE 802.11r (FT-BSS Transition) has lower packet loss (no more than 5%) compared to other combinations. The mobile node in our proposed combination takes less time to reassociate with

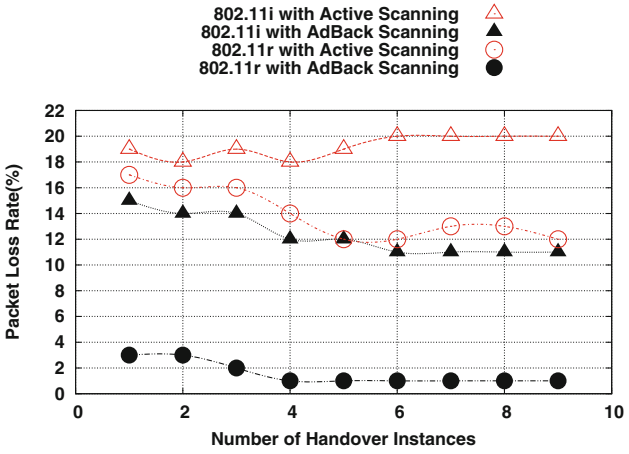


Fig. 3. Packet loss in percentage

the new RSU (approximately 35 ms) which reduces the number of packet losses. The packet loss is directly related to the handoff latency. Thus, our proposed mechanism outperforms. It has handover delay and packet loss of an acceptable QoS level for delay-sensitive applications.

6.2 Throughput Measurement Using Iperf Tool

To evaluate the performance in terms of throughput, we are using Iperf, network performance monitoring tool. The corresponding node works as a server and vehicular nodes as a client. The throughput measured through Transmission

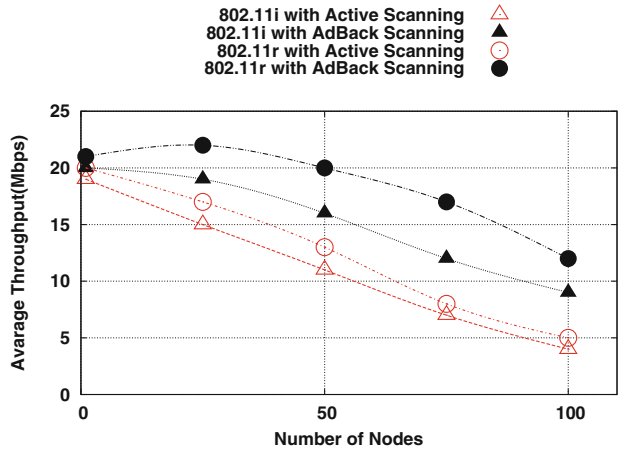


Fig. 4. Average throughput in Mbps

Control Protocol (TCP) tests in varying vehicular node density. A graph of the average throughput in Megabit per second (Mbps) versus vehicular node density shown in Fig. 4. We use the same test combinations and it can be seen that the disruption time due to these handover mechanisms affects performance. Although 802.11g mode is used, that supports a transmission speed of 54 Mbps, the throughput decreases significantly in the first method when the vehicular node density increases. The throughput in case of AdBack scanning is higher than the Active scanning. In the proposed combination of IEEE 802.11r with AdBack scanning, the handover execution is faster. It performs much better than the rest and throughput does not decrease rapidly in varying density. We can observe that throughput decreases when the number of vehicles increases because the wireless channel is shared among a large number of nodes.

7 Conclusion

In this work, we have proposed a new mechanism to maximize the quality of service for ITS applications and services in smart city Wi-Fi setup. In our proposed scheme, the collected information about the discovered APs during the periodic scan is cached. If RSSI drops to the defined threshold, it does not need to scan again, instead select potential AP from a cached neighbor list. This approach reduces the discovery delay drastically. Moreover, our scanning mechanism adapts to different mobility modes and does not require any modifications at the AP. In a smart-city highly secure Wi-Fi with WPA Enterprise (802.1X\EAP), reauthentication delay (inclusive of key-management) during each handover can cause a significant interruption to many services. The IEEE 802.11r FT over-the-DS reduces handover delay (due to reauthentication) by over 50% because the mobile node is already pre-authenticated in its network domain.

The WPA2 Enterprise (WPA2 802.1X/EAP) security has been used to provide authentication, privacy, integrity, and availability. This security standard is still considered the gold standard for wireless network security. The combination of our proposed AdBack scanning and IEEE 802.11r based fast reauthentication mechanism maximizes network throughput, minimizes handover latency and packet loss and complies with the QoS requirements for V2I applications mentioned in Sect. 1. This approach can be helpful for delay-sensitive applications such as VoIP and real-time services.

In this study, we did not compare the energy consumption, because vehicles do not suffer from power constraints like handheld devices. If a vehicle engine is running, it can power itself and always have sufficient energy. Most of the time, vehicles are either in the parking lot or driveway, during this period the proposed daemon running on OBU is energy efficient and will not trigger the scanning if it receives a better signal.

Our approach is simple in its implementation and does not require any change in AP side or installation of any additional server. As a future work, we are focusing on dynamic handoff management, threshold selection along with load balancing in a high mobility scenario of vehicular communication.

Acknowledgment. We are thankful to R. R. Fontes, one of the authors of [32] and developer of Mininet-WiFi who provided us help in solving the problems related to our experimental setup on Mininet-WiFi.

References

1. Consortium, C.V.S.C., et al.: Vehicle safety communications project: task 3 final report: identify intelligent vehicle safety applications enabled by DSRC. National Highway Traffic Safety Administration, US Department of Transportation, Washington DC (2005)
2. IEEE Std 802.11r/D01.0: Draft Amendment to Standard for Information Technology Telecommunications and Information Exchange Between Systems LAN/MAN Specific Requirements Part 11: Wireless Medium Access Control (MAC) and Physical Layer Specifications: Amendment 8: Fast BSS Transition
3. IEEE Std 802.11i: IEEE Standard for Wireless LAN Medium Access Control (MAC) and Physical Layer Specifications: Amendment 6: Medium Access Control Security Enhancements
4. Mishra, A., Shin, M., Arbaugh, W.: An empirical analysis of the IEEE 802.11 MAC layer handoff process. ACM SIGCOMM Comput. Commun. Rev. **33**(2), 93–102 (2003)
5. Shin, S., Forte, A.G., Rawat, A.S., Schulzrinne, H.: Reducing MAC layer handoff latency in IEEE 802.11 wireless LANs. In: Proceedings of the Second International Workshop on Mobility Management and Wireless Access Protocols, pp. 19–26. ACM (2004)
6. Bangolae, S., Bell, C., Qi, E.: Performance study of fast BSS transition using IEEE 802.11r. In: Proceedings of the 2006 International Conference on Wireless Communications and Mobile Computing, pp. 737–742. ACM (2006)
7. Park, S.-H., Kim, H.-S., Park, C.-S., Kim, J.-W., Ko, S.-J.: Selective channel scanning for fast handoff in wireless LAN using neighbor graph. In: Niemegeers, I., de Groot, S.H. (eds.) PWC 2004. LNCS, vol. 3260, pp. 194–203. Springer, Heidelberg (2004). https://doi.org/10.1007/978-3-540-30199-8_16
8. Sarma, A., Chakraborty, S., Nandi, S., Choubey, A.: Context aware inter-bss handoff in IEEE 802.11 networks: efficient resource utilization and performance improvement. Wireless Pers. Commun. **77**(4), 2587–2614 (2014)
9. Brik, V., Mishra, A., Banerjee, S.: Eliminating handoff latencies in 802.11 WLANs using multiple radios: applications, experience, and evaluation. In: Proceedings of the 5th ACM SIGCOMM Conference on Internet Measurement, IMC 2005, p. 27 (2005)
10. Jin, S., Choi, S.: A seamless handoff with multiple radios in IEEE 802.11 WLANs. IEEE Trans. Veh. Technol. **63**(3), 1408–1418 (2014)
11. Ramani, I., Savage, S.: SyncScan: practical fast handoff for 802.11 infrastructure networks. In: INFOCOM 2005, 24th Annual Joint Conference of the IEEE Computer and Communications Societies, Proceedings IEEE. vol. 1, pp. 675–684. IEEE (2005)
12. Chen, Y.S., Chuang, M.C., Chen, C.K.: DeuceScan: deuce-based fast handoff scheme in IEEE 802.11 wireless networks. IEEE Trans. Veh. Technol. **57**(2), 1126–1141 (2008)

13. Yoon, M., Cho, K., Li, J., Yun, J., Yoo, M., Kim, Y., Shu, Q., Yun, J., Han, K.: Adaptivescan: the fast layer-2 handoff for WLAN. In: 2011 Eighth International Conference on Information Technology: New Generations (ITNG), pp. 106–111. IEEE (2011)
14. Wu, T.Y., Obaidat, M.S., Chan, H.L.: Qualityscan scheme for load balancing efficiency in vehicular ad hoc networks (VANETs). J. Syst. Softw. **104**, 60–68 (2015)
15. Lee, J., Cho, S.-P., Kim, H.: Position based handover control method. In: Gervasi, O., Gavrilova, M.L., Kumar, V., Laganà, A., Lee, H.P., Mun, Y., Taniar, D., Tan, C.J.K. (eds.) ICCSA 2005. LNCS, vol. 3481, pp. 781–788. Springer, Heidelberg (2005). https://doi.org/10.1007/11424826_83
16. Montavont, J., Noel, T.: IEEE 802.11 handovers assisted by GPS information. In: 2006 IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob 2006), pp. 166–172. IEEE (2006)
17. Nicholson, A.J., Noble, B.D.: Breadcrumbs: forecasting mobile connectivity. In: Proceedings of the 14th ACM International Conference on Mobile Computing and Networking, pp. 46–57. ACM (2008)
18. Zhao, Y., Li, W., Lu, S.: Navigation-driven handoff minimization in wireless networks. J. Netw. Comput. Appl. **74**, 11–20 (2016)
19. Navda, V., Subramanian, A.P., Dhanasekaran, K., Timm-Giel, A., Das, S.: Mobisteer: using steerable beam directional antenna for vehicular network access. In: Proceedings of the 5th International Conference on Mobile Systems, Applications and Services, pp. 192–205. ACM (2007)
20. Balasubramanian, A., Mahajan, R., Venkataramani, A., Levine, B.N., Zahorjan, J.: Interactive wifi connectivity for moving vehicles. ACM SIGCOMM Comput. Commun. Rev. **38**(4), 427–438 (2008)
21. Deshpande, P., Kashyap, A., Sung, C., Das, S.R.: Predictive methods for improved vehicular wifi access. In: Proceedings of the 7th International Conference on Mobile Systems, Applications, and Services, pp. 263–276. ACM (2009)
22. Gañán, C.H., Reñé, S., Muñoz-Tapia, J.L., Esparza, O., Mata-Díaz, J., Alins, J.: Secure handoffs for V2I communications in 802.11 networks. In: Proceedings of the 10th ACM Symposium on Performance Evaluation of Wireless Ad Hoc, Sensor, and Ubiquitous Networks, pp. 49–56. ACM (2013)
23. Tabassam, A.A., Trsek, H., Heiss, S., Jasperneite, J.: Fast and seamless handover for secure mobile industrial applications with 802.11r. In: 2009 IEEE 34th Conference on Local Computer Networks, LCN 2009, pp. 750–757. IEEE (2009)
24. Martinovic, I., Zdarsky, F.A., Bachorek, A., Schmitt, J.B.: Measurement and analysis of handover latencies in IEEE 802.11i secured networks. In: Proceedings of the 13th European Wireless Conference (EW2007), Paris, France (2007)
25. Machań, P., Wozniak, J.: On the fast BSS transition algorithms in the IEEE 802.11r local area wireless networks. Telecommun. Syst. **52**(4), 2713–2720 (2013)
26. WPASupplicant: Bgscan. https://w1.fi/cgit/hostap/plain/wpasupplicant/wpa_supplicant.conf
27. Behrisch, M., Bieker, L., Erdmann, J., Krajzewicz, D.: Sumo-simulation of Urban mobility: an overview. In: Proceedings of SIMUL 2011, The Third International Conference on Advances in System Simulation, ThinkMind (2011)
28. Fontes, R.R., Afzal, S., Brito, S.H., Santos, M.A., Rothenberg, C.E.: Mininet-wifi: emulating software-defined wireless networks. In: 2015 11th International Conference on Network and Service Management (CNSM), pp. 384–389. IEEE (2015)
29. MininetWiFi: Discussion group. https://groups.google.com/forum/#!topic/mininet-wifi-discuss/jez7TA98eJQ

Aloe: An Elastic Auto-Scaled and Self-stabilized Orchestration Framework for IoT Applications

Subhrendu Chattopadhyay*, Soumyajit Chatterjee†, Sukumar Nandi‡, Sandip Chakraborty§
*‡ Department of CSE, IIT Guwahati, India
†§ Department of CSE, IIT Kharagpur, India

***Abstract*—Management of networked Internet of Things (IoT) infrastructure with in-network processing capabilities is becoming increasingly difficult due to the volatility of the system with low-cost resource-constraint devices. Traditional software-defined networking (SDN) based management systems are not suitable to handle the plug and play nature of such systems. Therefore, in this paper, we propose Aloe, an elastically auto-scalable SDN orchestration framework. Instead of using service grade SDN controller applications, Aloe uses multiple lightweight controller instances to exploit the capabilities of in-network processing infrastructure. The proposed framework ensures the availability and significant reduction in flow-setup delay by deploying instances near the resource constraint IoT devices dynamically. Aloe supports fault-tolerance and can recover from network partitioning by employing self-stabilizing placement of migration capable controller instances. The performance of the proposed system is measured by using an in-house testbed along with a large scale deployment in Amazon web services (AWS) cloud platform. The experimental results from these two testbed show significant improvement in response time for standard IoT based services. This improvement of performance is due to the reduction in flow-setup time. We found that Aloe can improve flow-setup time by around $10\% - 30\%$ in comparison to one of the state of the art orchestration framework.**

***Index Terms*—Network architecture, Fault-tolerance, Programmable network, software defined network**

I. Introduction

The rapid proliferation of Internet-of-Things (IoT) has made the network architecture complicated and difficult to manage for service provisioning and ensuring security to the end-users. Simultaneously, with the advancement of edge-computing, in-network processing (also known as fog computing) and platform-as-a-service technologies, end-users consider the network as a service platform for the deployment and execution of myriads of diverse applications dynamically and seamlessly over the network. Consequently, network management is becoming increasingly difficult in today's world with this complex service-oriented platform overlay on top of the inherently distributed TCP/IP network architecture. The concept of software-defined networking (SDN) has gained popularity over the last decade to make the network management simple, cost-effective and logically centralized, where a network manager can monitor, control and deploy new network services through a central controller. Nevertheless, edge and in-network processing over an IoT platform is still challenging even with an SDN based architecture [1].

The primary requirements for supporting edge and in-network processing over a networked IoT platform are as follows: (1) The platform should be agile to support rapid deployment of applications without incurring additional overhead for in-network processing [2]. This also ensures scalability of the system [3]. (2) Many times, in-network processing requires dividing a service into multiple microservices and deploying the microservices at different network nodes for reducing the application response time with parallel computations [4]. However, such microservices may need to communicate with each other, and therefore the flow-setup delay from the in-network nodes need to be very low to ensure near real-time processing. (3) The percentage of short-lived flows are high for IoT based networks [5]. This also escalates the requirement for reducing flow-setup delay in the network. (4) Failure rates of IoT nodes are in-general high [6]. Therefore, the system should support a fault-tolerant or fault-resilient architecture to ensure liveness.

Although SDN supported edge computing and in-network processing have been widely studied in the literature for the last few years [1], [7] as a promising technology to solve many of the network management problems associated with large-scale IoT networks, they have certain limitations. First of all, the SDN controller is a single-point bottleneck. Every flow initiation requires communication between switches and the controller; therefore, the performance depends on the switch-controller delay. With a single controller bottleneck, the delay between the switch and the controller increases, which affects the flow-setup performance. As we mentioned earlier that the majority of the flows in an IoT network are short-lived flows, the impact of switch-controller delay is more severe on the performance of short-lived flows. To solve this issue, researchers have explored distributed SDN architecture with multiple controllers deployed over the network [8]. However, with a distributed SDN architecture, the question arises about how many controllers to deploy and where to deploy those controllers. Static controller deployments may not alleviate this problem, as IoT networks are mostly dynamic with a plug-and-play deployment of devices. Dynamic controller deployment requires hosting the controller software over commercially-off-the-shelf (COTS) devices and designing methodologies for

Email:subhrendu@iitg.ac.in*,soumyachat@iitkgp.ac.in†,sukumar@iitg.ac.in‡, sandipc@cse.iitkgp.ac.in§

This work is partially supported by TCS India, Microsoft India, LRN Foundation and CNeRG IIT Kharagpur.

controller coordination which is a challenging task [9]. The problem is escalated with the objective of developing a fault-tolerant or fault-resilient architecture in a network where the majority of the flows are short-lived flows.

In contrast to the existing architectures, the novelty of this paper is as follows. We integrate an SDN control plane with the in-network processing infrastructure, such that the control plane can dynamically be deployed over the COTS devices maintaining a fault-tolerant architecture. This has multiple advantages for an IoT framework with in-network processing capabilities: (a) The distributed controller approach ensures that there is no performance bottleneck near the controller. (b) The flow-setup delay is significantly minimized because of the availability of a controller near every device. (c) The fault-tolerant controller orchestration ensures the liveness of the system even in the presence of multiple simultaneous devices or network faults. To achieve these goals, we first design a distributed, robust, migration-capable and elastically scalable control plane framework with the help of docker containers [10] and state-of-the-art control plane technologies. The proposed control plane consists of a set of small controllers, called the micro-controllers, which can coordinate with each other and help in deploying new applications for in-network processing. The container platform helps in installing these micro-controllers on the COTS devices; a container with a micro-controller can be seamlessly migrated to another target device if the host device fails, yielding a fault-tolerant architecture. In addition to this, the deployment mechanism for the micro-controllers ensure elastic auto-scaling of the system; the total number of controllers can grow or sink based on the number of active devices in the IoT network. We develop a set of special purpose programming interfaces to ensure fault-tolerant elastic auto-scaling of the system along with intra-controller coordinations. Finally, we design a set of application programming interfaces (API) over this platform to ensure language-free independent deployment of applications for in-network processing. Combining all these concepts, we present Aloe[1], a distributed, robust, elastically auto-scalable, platform-independent orchestration framework for edge and in-network processing over IoT infrastructures.

We have implemented a prototype of Aloe using state-of-the-art SDN control plane technologies and deployed the system over an in-house testbed and a 68-node Amazon web services platform. The in-house testbed consists of 10 nodes (Raspberry Pi devices) with Raspbian kernel version 8.0. As mentioned, we have utilized docker containers to host the distributed control plane platform. We have tested Aloe with three popular applications for in-network IoT data processing – (a) A web server (simple python based), (b) a distributed database server (`Cassandra`), and (c) a distributed file storage platform (`Gluster`). We observe that Aloe can reduce the flow-setup delay significantly (more than three times) compared to state-of-the-art distributed control plane technologies while boosting up application performance even in the presence of multiple simultaneous faults.

[1]Aloevera: A plant known for its healing property.

II. Related Work

Traditional single controller architecture is not suitable for IoT infrastructure, where the network is dynamic and failure prone. One way to address such a problem is to deploy a distributed control plane. However, existing distributed control planes are not efficient for handling large-scale IoT systems that require in-band control. ONIX [11] and ONOS [12] are two popular distributed control plane architectures. ONIX uses a distributed hash table (DHT) data store for storing volatile link state information. On the other hand, ONOS uses NoSQL distributed database and distributed registry to ensure data consistency. Although both of them can scale easily and show a significant amount of fault-resiliency, they require high end distributed computing infrastructure for execution. Deployment of such infrastructure increases the cost of IoT deployment and leads to performance degradation of IoT services which rely on short flows due to the increase in the number of controller consultation. On the other hand, IoT devices require in-band control as most of them have limited network interfaces. Therefore, a disruption in IoT link can have a severe impact on multiple IoT nodes due to disconnection from the control plane. Similar problems can be observed in case of Kandoo [13] and Elasticon [14].

To avoid these challenges DIFANE [15], DevoFlow [16] and BLAC [17] design control plane by utilizing data plane services. DIFANE and DevoFlow use special purpose switches which can take decisions in its local neighborhood in the absence of the controller. However, this requires switch-level modifications which may not be possible for every hardware components. On the other hand, BLAC uses a controller scheduling mechanism to dynamically scale the control plane to accommodate the need the system. However, BLAC increases the flow setup time and not suitable for real-time IoT applications.

In SCL [18], the authors have developed a coordination layer which can provide consistent updates for a single image, lightweight controllers deployed in an in-band fashion. However, SCL uses two-phase commit mechanism for consistency preservation, which incurs high latency. Moreover, SCL is not suitable for IoT services as it assumes the existence of robust channels among switch and controllers, which is not suitable for low-cost and resource-constrained networked IoT systems.

III. Components of Aloe

The Aloe orchestration framework exploits the capabilities of the in-network processing architecture over an IoT based platform where devices work mostly in a plug-and-play mode. The main components of the architecture are shown in Fig. 1. It can be noted here that the proposed architecture does not bring new hardware or software platforms at its base; instead, we utilize the available COTS hardware and open-source software suites to design this entire architecture. Our objective is to design an orchestration platform that can be developed with market-available components while integrating

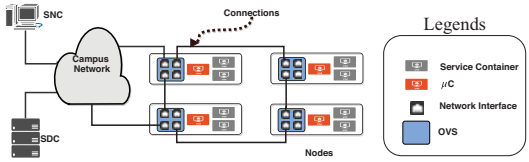


Fig. 1: Components of Infrastructure

innovations in the design such that the shortcomings of the existing systems can be mitigated. We discuss the individual components and their functionalities in this section.

A. Infrastructure Nodes

The networking equipment and devices are considered as the infrastructure nodes. Therefore, nodes are essentially embedded and resource-constraint devices like smart-gateways, smart routers, smart IoT monitoring devices, etc. These devices participate in communication and provide in-network processing platforms for lightweight services by utilizing residual resources. We consider that these nodes are either SDN-supported or can be configured with open-source software platform like *open virtual switch* (OVS) to make them SDN capable.

We use containerized platforms like docker [10] to offload services in the IoT platform for in-network processing. The containerized service deployment helps in supporting service isolation and makes the architecture failsafe by supporting live migration of containers. Further, containers reduce a programmer's overhead for service delegation and cost of deployment, as the same device can be used for in-network processing of IoT applications along with execution of custom networking services.

B. Service Deployment Controller

To identify the resource requirement and delegation of the services which require in-network processing, we use a centralized service deployment controller (SDC). The SDC periodically monitors the resource consumptions of the nodes. Once a new service is ready for deployment in the system, SDC identifies the schedules in which the services can be executed by the nodes without violating resource demands from individual services. Once the schedule is generated, the SDC is responsible for delegating the services based on the schedule. It can be noted that the load of an SDC is much less compared to the network management controller. Therefore we maintain a single instance of SDC in our system.

C. Super Network Controller

Network management in an IoT system is non-trivial due to the diversified inter-service communication requirements and the dynamic nature of the network. Aloe uses a two-layer approach. We deploy a high availability super network controller (SNC) at the first layer, which is responsible for storing persistent network information, like routing protocols, quality of service (QoS) requirements, the periodicity of statistic collection from nodes, etc. An SNC also manages an access control list (ACL) to provide necessary security to the infrastructure nodes.

D. Micro-Controllers

Although super controllers are highly available, an IoT infrastructure has a time-varying topology due to the use of the resource constraint devices and the devices being plug-and-play most of the times. Therefore, the use of a centralized controller cannot achieve fault-tolerance (failure of infrastructure nodes) and partition-tolerance (failure of network links resulting in network partitions). On the other hand, unlike SDC, SNC needs to be consulted by the nodes each time a new flow enters the system. This increases the communication overhead and flow initiation delay which also affects the performance of the services deployed in the infrastructure. Therefore, Aloe uses a second layer of network controllers named as *"micro-controllers"* (μC).

μCs are lightweight SDN controllers. A μC stores volatile link layer information of a small group of nodes placed topologically close to it. Thus a μC maintains information consistency by minimizing the delay between the governing μC and the nodes managed by it. The SNC can aggregate these statistics via `REST API` queries from the μC. Based on the changing QoS of the services, network service provisioning can be achieved in the μC via the same `REST API`. Based on the configuration of the SNC, a μC collects statistics from individual OVS modules of the nodes. Thus a μC can achieve a fine-tuned network control for the infrastructure nodes.

However, deployment of μCs in nodes might also create network partitioning issue. To avoid such an undesirable scenario, Aloe uses a novel approach where the μCs are encapsulated inside a container and deployed as a service inside the infrastructure nodes itself. Thus Aloe supports *μC as a Service* (μCaaS) which ensures fault-tolerance of the system. μC containers can be migrated to a target node quite easily with the help of the live-migration technique of a container when the host node fails. Aloe ensures that a set of μCs is always live in the system maintaining the requirements for minimized switch-controller delay. On the other hand, a μC container can be customized depending on the available capacity of the nodes and resource consumptions by the controller applications. It can be noted that this μC architecture is different from existing distributed SDN controller approaches, such as DevoFlow [16] and SCL [18], which require switch-level customization. μCs can run over the existing COTS devices without any requirement for switch-level modifications.

IV. Design of Aloe Orchestration Framework

This section discusses the Aloe orchestration framework by highlighting various functional modules of Aloe and their working principles. Finally, we develop a set of APIs for language-independent and robust deployment of applications over the Aloe framework. The various functional modules of Aloe are shown in Fig. 2; the detailed description follows.

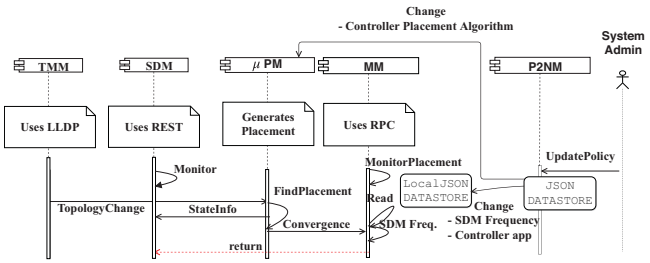


Fig. 2: Aloe function modules and their interactions

A. Aloe Functional Modules

The proposed framework consists of four node-level modules and one SNC-level module. The node-level modules run inside the infrastructure nodes and decide the topology and service parameters that need to be synchronized across various nodes. These modules collaborate with each other to take distributed decisions in a fault-tolerant way. It can be noted that in Aloe, infrastructure nodes are mutable and they can convert themselves as a μC if required. An interesting feature of Aloe is that this decision mechanism is executed in a pure distributed way, preserving the safety and liveness of the system in the presence of faults. The functionalities of various modules are as follows.

1) Topology Management Module (TMM): We design Aloe as a plug-and-play service, where an Aloe-supported IoT device can be directly deployed in an existing system for flexible auto-scaling support. The TMM initializes the Aloe framework on a newly deployed node. The tasks of the TIM are as follows – (i) identify the nodes in the neighborhood, and (ii) determine whether an Aloe service is running in that node. An Aloe service is of two types – (a) μC service, and (b) user application service. To find out the active nodes in the neighborhood, TMM uses *Link Layer Discovery Protocol* (LLDP). We assume that each Aloe service deployed in the IoT cloud uses a unique predefined port address. TIM queries about the services in the local neighborhood via issuing a `telnet open port` requests. Apart from the initialization, this module is invoked whenever a node/link failure or μC failure event is detected.

2) State Discovery Module (SDM): In case of a node or a link failure after the initialization through TMM, there is a possibility that the infrastructure nodes get disconnected from the μC. To identify such a scenario, Aloe maintains various state variables for each node as follows. (i) *Controller State* (CTLR): This state variable decides whether a node is in a `general` (does not host a μC service), `μC` (hosts a μC) or `undecided` (an intermediate state between the general state and the μC state) state. (ii) *Priority* (PRIO): This state variable is required only if the node is undecided and denotes the priority of the node for becoming a μC. The states associated to nodes are kept and managed by the nodes themselves. However, a node can access a copy of the states from its neighbor to decide its state. SDM is responsible for accumulating the state information collected from the neighbors. SDM uses `REST` for this purpose. Once a failure event occurs, TMM invokes the SDM. SDM keeps on executing periodically until the node finds at least one μC in its neighborhood. The periodicity of the execution of this module is dependent on the link delay. For implementation purpose, we consider the periodicity as the largest delay observed to fetch data from a neighbor.

3) μC Placement Module (μPM): Based on the neighbor states collected through SDM, every node independently determines whether it needs to launch a μC service. This is done through the μPM module. We consider the nodes as the vertices of a graph where the edges determined by the connectivity between two nodes, and place the μC services to the nodes that form a *maximal independent set* (MIS) on that graph. An MIS based μC placement ensures that there would be a μC at least in one-hop distance from each node, which can take care of the configurations and flow-initiations for the application services running on that node. As we have claimed earlier and will show in § VI that the μCs utilized in Aloe are significantly light-weight but efficient for performing network and service management activities. Therefore the total overhead due to MIS based μC placement is not significant. For identification of a suitable set of μC capable nodes, we develop a distributed randomized MIS algorithm given in Algorithm 1. The novelties of this algorithm are as follows. (1) **Randomized**: The algorithm selects different nodes at different rounds, ensuring that the load for μC service hosting is distributed across the network and does not get concentrated on some selected nodes. (2) **Bounded set**: The number of deployed μCs are always bounded based on the total number of nodes in the network. (3) **Self-stabilized**: The algorithm is self-stabilized and converges in linear time (the proof is omitted due to space-constraint), ensuring fault-tolerance of the system under single or multiple simultaneous faults until complete network partition occurs. Intuitively, we can see that; a node cannot retain its status as `undecided` forever. In a neighborhood, if it finds another μC then it changes to `general` (Line 24), otherwise, it competes with other nodes having `undecided` status in a series of tiebreaker rounds by choosing a random `PRIO` value (Line 25). Until one of the nodes receives a unique maximum priority in the neighborhood and gains `μC` status (Line 29), the random trials continue (Line 31). It can be shown that the expected number of such trials for a node is less than thrice the number of competing nodes (proof omitted due to space constraint). Therefore, the Algorithm 1 always converges in linear time.

4) μC Manager Module (μMM): Once a node decides its state through μPM, the μMM module initiates the μC service on the selected nodes and establishes a controller-switch relationship between the μC and the nodes with *general* state in the one-hop neighborhood. As we mentioned earlier, a μC is initiated as a containerized service over the node designated for hosting a μC by the μPM algorithm. For a node with *general* state, this process may involve changing of

Algorithm 1: μPM Controller Placement Algorithm

```
1  Function Trial():
2      PRIO← Rand();
       ;                                  // Breaks priority ties
3      return;
4  Function NeighborμC():
5      if Another μC in one-hop neighborhood then
           ;                              // If μC in neighborhood
6          return true
7      else
8          return false
9  Function UMPriority:
       /* If node has unique maximum priority     */
10     if PRIO of this node > maximum PRIO in neighborhood then
11         return true
12     else if PRIO of this node = maximum PRIO in neighborhood then
13         return false
14     else
15         return None
16 Function Main():
17     while state change detected do
18         if CTLR=general & NeighborμC()=false then
               // No μC in neighborhood
19             CTLR← undecided;
20             Trial();
               // Initialize priority
21         else if CTLR=μC & NeighborμC()=true then
               /* Two μCs are adjacent                */
22             CTLR← general;
23         else if CTLR=undecided & NeighborμC()=true then
               /* μC found in neighborhood            */
24             CTLR← general;
25         else
26             if UMPriority()=None then
                   /* Executor is not maximum         */
27                 continue;
28             else if UMPriority()=true then
                   /* Unique maximum priority.        */
29                 CTLR← μC;
30             else
                   /* Maximum but not unique priority
                      */
31                 CTLR← undecided;
                   /* Next round of trial starts      */
32                 Trial();
33     return
```

controller services from one μC to another μC, which requires the reestablishment of the controller-switch relationship. For this purpose, the SDN flow tables need to be migrated from the old μC to the newly associated μC. The flow table migration mechanism is specific to the SDN controller software used, and therefore, we discuss it in §V.

5) PushToNode Module (P2NM): Along with fault-tolerance, Aloe supports rapid deployment and runtime customization of the system. To implement this feature, we develop P2NM. Unlike the rest of the modules, P2NM is centralized and/or deployed in the SNC. It provides an interface for monitoring and changing the policy level information for the μC at runtime which is useful for system administrators. Aloe supported policy level information include (i) ACLs, (ii) controller application to be executed in the μC, (iii) routing protocols running in the μC, and (iv) SDM update frequency. Apart from the specified policies, Aloe also gives freedom to its user to customize the Aloe modules itself. This feature is achieved by developing a set of Application Programmer's Interfaces (API) as discussed next.

B. Application Programmer's Interfaces (API)

The primary objective of this orchestration framework is to deploy the *controller as a service* to the in-network processing infrastructure in the form of a μC. There are some significant differences between a user application service and a μC service, which makes the deployment of the later non-trivial. Unlike user application services, μC services should not act location transparently. A location transparent deployment of μC might allocate all the μCs in the same node, which can degrade the network performance of the infrastructure. Therefore, during the design of Aloe, we consider the extendibility of this work. Many of the implemented functionalities of this framework can be reused as API for distributed controller application development. For ease of understanding, we only provide the python sample programs here. However, all the APIs can be invoked as a `script` over the SNC using P2NM.

1) Topology Monitor: Using this python API, Aloe can detect a topology change event (`TopologyMonitor()`) and take actions accordingly. This API can also be used for general purpose routing application, as given in the following code.

```
''' Find shortest path between dpidS and dpidD'''
import networkx as nx
G=TopologyMonitor()
path_dpid_list=nx.algorithms.shortest_path(G,"delay"
    )
```

Listing 1: Topology Change Detector

2) Distributed State Inspector: We develop this API to observe the state of the nodes (`getNeighborStates()`), which helps in developing new placement algorithms for μPM. This API relies on a remote procedure call (`rpc`).

```
''' Find max priority amongst neighbor'''
import json
states=getNeighborStates()
maxPrioUndecided=max([v["Prio"] for v in states.
    values() if state["CTLR"]=="undecided"])
```

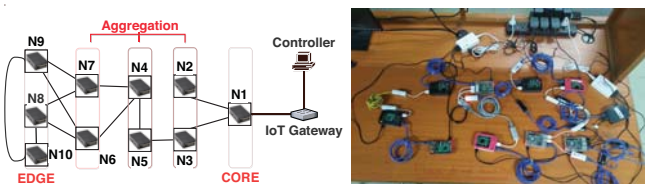
Listing 2: Distributed State Inspector

3) Find Node Services: The framework requires to identify the deployed services (`getNeighborServices()`) to enforce service level policies. We provide a python API to ease this task. The following example can be used for selective service blocking ACL.

```
'''Service blocking (ACL)'''
import json,os
services=getNeighborServices()
bport=blocking_port
if(blocking_port in service["dpid"]):
 os.system("ovs-ofctl add-flow match:src=dpid,
    tcp_port=bport action:drop")
```

Listing 3: Find Node Services

Next, we discuss the details of the Aloe implementation as a general orchestration framework.



(a) Topology (b) In Action

Fig. 3: Testbed

TABLE I: Wilcoxon Rank Sum Test (↑indicates μC in top header consumes less resources, ←indicates μC in left header consumes less resources, **X** indicates the choice is undetermined)

CPU				
μC	No μC	Ryu	Zero	ODL
No μC		←	←	←
Ryu	< 0.0001		←	←
Zero	< 0.0001	< 0.0001		←
ODL	< 0.0001	< 0.0001	< 0.0001	
Memory				
No μC		←	X	←
Ryu	< 0.0001		X	←
Zero	> 0.03	> 0.01		←
ODL	< 0.0001	< 0.0001	< 0.0001	
CPU Temperature				
No μC		X	←	←
Ryu	> 0.39		←	←
Zero	< 0.0001	< 0.0001		↑
ODL	< 0.0001	< 0.0001	< 0.0001	

V. Aloe Implementation

We have implemented Aloe as a middleware over Linux kernel with the integration of open-source technologies, like docker containers, various SDN controllers, and `REST` based communication modules. We first discuss the implementation environment that we utilized, followed by a brief description of two different implementation aspects.

A. Environmental Setup

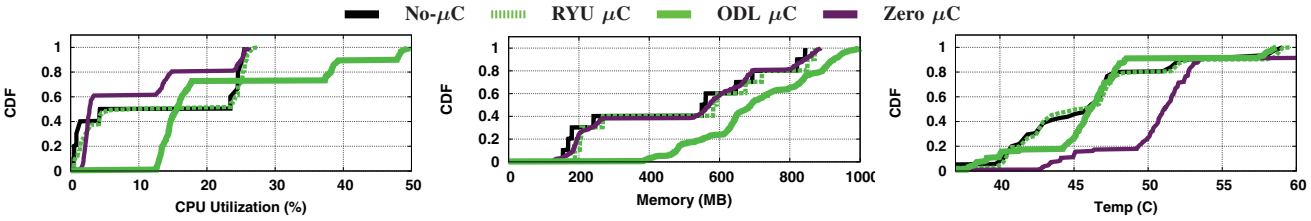
To analyze the performance of Aloe, we have deployed an in-house testbed using the topology given in Fig. 3a (Fig. 3b shows the live-snapshot of the testbed). The nodes in the testbed are Raspberry Pi version 3 Model B, which are configured with Raspbian 8.0 operating system with kernel version 4.4.50-v7+. The nodes are connected via multiple 100Mbps USB-to-Ethernet adapter-edges representing the physical Ethernet links among the nodes. We use Linux `tc` to configure each link to use 5Mbps of bandwidth and added $100ms$ of propagation delay to match real life IoT deployment specification. Further, to analyze the scalability of Aloe, we have also deployed Aloe in a large-scale 68-node testbed using Amazon Web Services (AWS). For this purpose, we consider a sub-topology from `rocketfuel` [19] topology which consist 68 nodes. The nodes in the topology are deployed using 18 AWS *nano* instances (1 vCPU and 512 MB RAM) and 50 AWS *micro* instances (1 vCPU and 1 GB RAM). The AWS nodes are configured with Ubuntu 16.10 operating system with Debian kernel version 4.4.0. To emulate edges between the nodes, we use the *Layer 2 Tunneling Protocol* (l2tp) between the AWS instances. Every infrastructure node, both in the testbed and in the AWS, are configured with *open virtual switch* (OVS).

B. Implementation Aspects

Here we discuss two important implementation aspects of Aloe – (i) flow-table consistency preservation during μC migration, and (ii) choice of controller service for μC implementation. Here, we discuss these two aspects in details.

1) Migration of μC and consistency preservation: A change in a policy level parameter requires a migration of the flow tables from the old μC instances to the new instances of the μC. Similarly, after a μPM execution, there might be a need for change in node-controller association. To implement such functionality, we have implemented a `rpc` and `REST` based API (`changeCtlr()`) which can dynamically change a switch's allegiance towards a μC. `changeCtlr()` forces the node to invoke a "`controller re-association request`" to the target μC with its previous μC address. After receiving a "`controller re-association request`", the target controller invokes migration of flow entries from the previously assigned μC. During the migration procedure, it is important to keep track of the previous state informations. To ensure the consistency, Aloe preserves snapshots of the μC flow table entries by sending `REST` queries to the μCs before the migration process starts. To make the migration process lightweight, the container instance is not transferred from one node to another node; instead, the source node container is killed, and a new container is invoked at the destination node via `rpc`. In case of a network partitioning between the previous μC and the target μC, the target μC obtains the copy of flow table from the requester node itself. In this way, the μMM preserves weak consistency in the system.

2) Choice of controller service for μC: The efficiency of Aloe is dependent on the efficiency of the choice of a controller service for the μC. Deployment of a heavy-weight controller can over-consume resources of the nodes; moreover, one μC is only responsible for managing a small set of nodes. Therefore, we target to opt for a light-weight μC for Aloe. In order to identify a suitable controller platform for μC, we compare a set of existing SDN controller services like "`Open Day light (ODL)`" [20], "`ONOS`" [12], "`ryu`" and "`Zero`" in our in-house testbed in terms of theirs resource utilization. Amongst these controllers, "`ONOS`" requires high memory consumption ($> 500MB$) which creates an instability in the docker environment. Further, we have observed that approximately 32% times, "`ONOS`" fails to execute over the testbed nodes due to unavailability of sufficient virtual memory. Therefore, we report the performance of the controllers other than "`ONOS`". The performance is reported based on three major system parameters – CPU utilization, memory utilization and CPU temperature variation. In Fig. 4a, we provide the comparison of the performance of the competing



(a) CPU Utilization (b) Memory Utilization (c) CPU Temperature Variation

Fig. 4: Resource Utilization Comparison of Controller Applications

TABLE II: Wilcoxon Rank Sum Test conclusions with p-values over response time of different applications:
X=Inconclusive, ✓=*Aloe* better, •=In band better

Services / Failure	Cassandra	HTTP	Gluster
0	X (>0.11)	X (>0.20)	•(<0.0001)
1	✓(<0.0001)	X (>0.04)	•(<0.0001)
2	✓(<0.0001)	✓(<0.0001)	X (>0.39)
3	✓(<0.0001)	✓(<0.0001)	✓(<0.01)

controllers in terms of CPU utilization. The plot suggests that approximately 30% "ODL" μCs use more than 30% of the CPU utilization. In comparison to that, around 40% "Zero" μCs use 15% of the CPU utilization. In Fig. 4b, we observe that almost 80% "ODL" μCs use more than $600MB$ of memory space. All the other controllers show lower memory utilization. Fig. 4c shows the variation in CPU core temperature while executing different types of controller services. The consolidated pair-wise comparison of the controllers are provided in Table I (upper right triangle in blue color). The notation **X** signifies that the comparison cannot be obtained. On the other hand ↑suggests that the μC listed in the top header consumes less amount of resources. We use ←to denote the higher efficiency of the μC application mentioned in the left header. To ascertain the comparison, we perform a statistical hypothesis testing using non-parametric, one-tailed Wilcoxon rank sum test ($\alpha = 0.01$) [21]. Our alternative hypothesis assumes that the mean resource consumptions and the core temperature will be higher than the one in the normal case. The left lower triangular part of Table I (given in green color) signifies the p-values obtained from the rank test. Based on our observation from Table I, we choose "Zero" as our choice of μC in testbed and AWS.

With this implementation, we evaluate the performance of Aloe, as discussed in the next section.

VI. Evaluation

We have tested the performance of Aloe with three different categories of standard applications which are common and useful for a networked IoT based system – (i) HTTP service (Python `SimpleHTTPServer`): used for bulk data transfer via web clients, (ii) distributed database service (`Cassandra`): for data-driven applications, and (iii) distributed file system service (`Gluster`): used for file sharing and fault-tolerant file replication over a distributed platform. We further compare the performance of Aloe with BLAC [17], a distributed SDN control platform. To emulate realistic fault models in the system, we have injected the faults using Netflix *Chaos Monkey* fault injection tool. We have taken the measurements under all possible fault combinations in the testbed and 100 different random fault combinations in AWS.

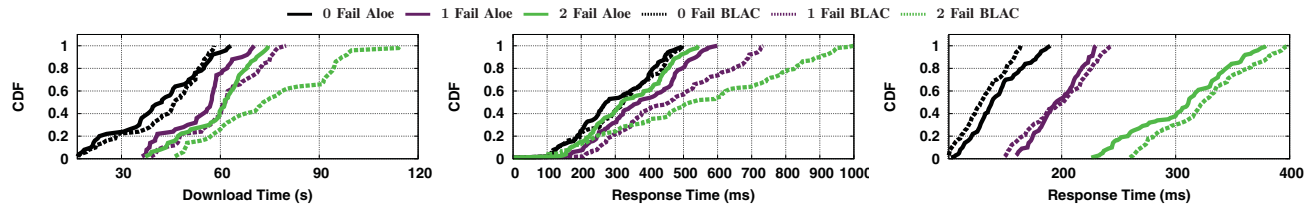
A. Application Performance

Fig. 5a compares the download time of a 512MB file hosted using the HTTP service under the influence of both BLAC and Aloe over the in-house testbed. The results are obtained by varying all possible source-destination pairs in the topology. We observe that, even though Aloe results in higher download time compared to BLAC when there is no failure in the system, the performance improves rapidly in the presence of link outage. While injecting failure, we observe that approximately 30% connections are timed-out while operating under the governance of the BLAC controller. However, Aloe reduces such flow termination[2] ($< 5\%$ connection time-out for Aloe). To compare the differences of the nature of the results for each service, we performed a Wilcoxon rank sum test. The p-values and conclusion from the test is summarised in Table II.

Fig. 5b shows the response time of `Cassandra` search queries. Here, we observe a significant difference in the characteristics of the plots due to the nature of the service. Unlike `HTTP`, `Cassandra` utilizes short flows to fetch query results. Therefore, we observe that Aloe provides a significant improvement in the query response time. However, in case of `Gluster`, Aloe performance is marginally poor compared to BLAC until there are 3 link failures (Fig. 5c). `Gluster` flows are short-distant flows, usually within one-hop. The flow-setup delay is almost negligible for a one-hop flow. Therefore the μC deployment overhead of Aloe is more when the number of failures is less.

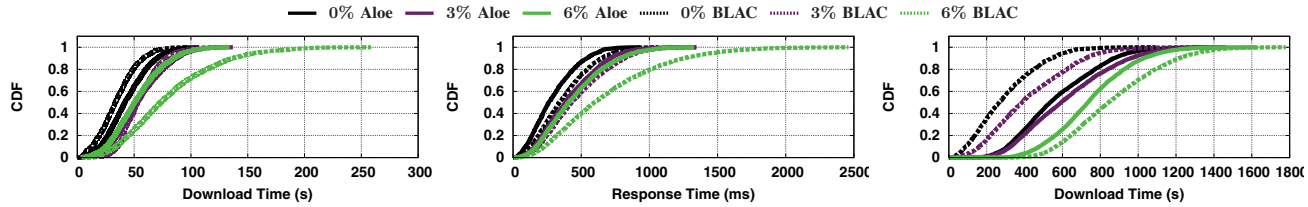
Similar behaviors are observed in the large-scale deployment of Aloe in the AWS cloud. In Figs. 6a and 6c, `HTTP` and `Gluster` response times show similar characteristics as observed in the testbed. In the case of `Cassandra` (Fig. 6b), all the cases perform significantly better than BLAC. From these observations, we conclude that Aloe performs significantly better for the services that generate long-distant mice flows (like database synchronization). For a long-distant flow, the flow setup delay is high, which gets further affected by

[2]Only in such cases, where the network is partitioned



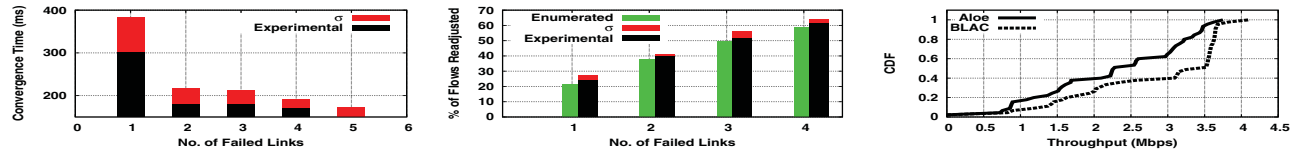
(a) HTTP Server (b) Cassandra (c) Gluster

Fig. 5: Comparison of response time of services obtained from testbed: Average percentage improvement of Aloe – (a) HTTP server: 4% (0-fail), 11% (2-fails), 21% (3-fails), (b) Cassandra: 9% (0-fail), 26% (2-fails), 37% (3-fails), (c) Gluster: -8% (0-fail), 0.1% (2-fails), 6% (3-fails)



(a) HTTP Server (b) Cassandra (c) Gluster

Fig. 6: Comparison of response time of services obtained from AWS cloud: Average percentage improvement of Aloe – (a) HTTP server: -2% (0-fail), 0.1% (2%-fails), 34% (6%-fails), (b) Cassandra: 20% (0-fail), 21% (2%-fails), 34% (6%-fails), (c) Gluster: -12% (0-fail), -6% (2%-fails), 14% (6%-fails)



(a) Avg. Aloe Convergence Time (b) Avg. No. Flow Restoration (c) Throughput comparison

Fig. 7: Testbed: Effect of failure on Aloe performance (σ= standard deviation)

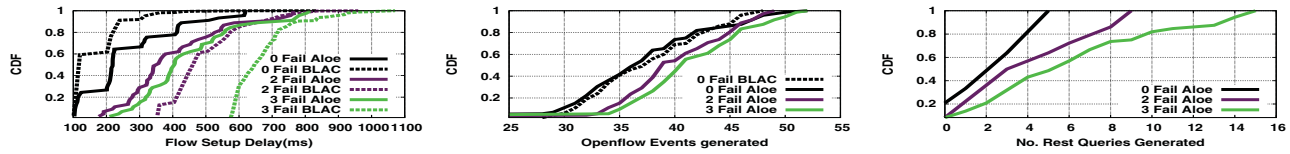
link failures. As a consequence, Aloe performance is better for failure-prone systems, like IoT clouds, as the flow-setup delay gets increased with the recovery time due to a failure.

B. Dissecting Aloe

Aloe flow-setup time is dependent upon the convergence time of μPM and the path restoration time. Fig. 7a shows the distribution of the average convergence time of Aloe in the presence of failure. We have an interesting observation here that as the number of simultaneous failures increases, the convergence time drops. This can be explained as follows. Let us consider two different faults. If the two faults are at two different sides of the network, then two waves of μPM starts executing simultaneously from two different ends of the network. These two waves get diffused in the network and meet in the middle of the network at convergence. That way, multiple faults create multiple such μPM waves in the network in parallel, and as these individual waves need to deal with a smaller part of the network, they converge quickly.

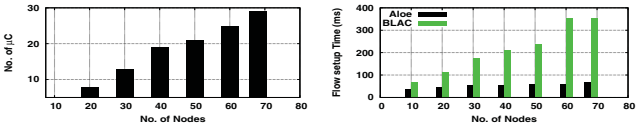
The convergence phase is followed by the path restoration due to a change of the controller positions in case of a failure. To identify the performance of the path restoration, we measure the average number of flow adjustments done by the framework. The number of flow adjustment depends on the topology and the locations of the failed links. Therefore, to compare the result, we provide the enumerated number of flow adjustment required for all possible cases of failures in Fig. 7b. We observe that the experimental observations closely match with the results obtained by enumeration. Further, these metrics have a direct impact on the flow-setup delay. To understand the effect of these factors, we compare the flow-setup delay for BLAC control plane and Aloe. To identify the flow-setup delay, we use `ping` to transfer a single `ICMP` packet. Fig. 8a shows that although Aloe marginally increases the flow setup-delay in the absence of a failure, it provides quick flow-setup when multiple faults occur in the network.

We observed that the overhead of distributed μC in Aloe is responsible for the increase in the flow-setup delay during the no-failure scenario. However, it is difficult to compare the exact overhead of the BLAC control plane and Aloe due to the difference of the nature the overhead. We measure the overhead regarding two different factors. Fig. 8b shows the comparison between BLAC and Aloe regarding the number of `openflow events` generated over a period of 100s. However, Aloe additionally generates `REST` queries to support



(a) Flow Setup Delay:Measured using `ping` (b) Control Plane Overhead (c) `REST` queries generated in Aloe

Fig. 8: Testbed: Flow setup delay and overhead comparison



(a) No. of μC (b) Avg. Flow Setup Time

Fig. 9: AWS: Effect of Scaling

inter-controller communication. Fig. 8c depicts the number of `REST` queries generated in Aloe. Due to these overheads, the Aloe incurs higher communication overhead than the BLAC control plane. However, due to the significant reduction in flow-setup time, Aloe ensures better flow throughput than BLAC, as shown in Fig. 7c.

Although Aloe incurs communication overhead, Aloe ensures a significant drop in the average flow-setup delay. To limit the flow-setup delay, Aloe provides elastic auto-scaling by increasing the number of μC instances to guarantee that each node can find a μC in its neighborhood. Fig. 9a shows the average number of μC instances when the network scales, as obtained from the AWS implementation. The effect of elastic auto-scaling is shown in Fig. 9b which indicates that the flow-setup delay only increases marginally in comparison to the BLAC controller, which incurs a significantly high flow-setup delay as the number of nodes in the network increases.

VII. Conclusion

In this paper, we present Aloe, an orchestration framework, for IoT in-network processing infrastructures. Aloe uses docker container to support lightweight migration capable in-band controllers. This design choice helps Aloe to provide elastic auto-scaling while keeping flow setup time under control. Aloe provides controller as a service to exploit in-network processing infrastructure and supports fault and partition tolerance. The performance of Aloe has been tested thoroughly using two real testbeds and compared with a very recent orchestration framework (BLAC). The results indicate a significant improvement in response times for distributed IoT services.

References

[1] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: a survey, use cases, and future directions," *IEEE Comm. Surveys & Tutorials*, pp. 2359–2391, 2017.

[2] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *SIGCOMM Comput. Commun. Rev.*, pp. 27–32, 2014.

[3] M. Chiang, S. Ha, I. Chih-Lin, F. Risso, and T. Zhang, "Clarifying fog computing and networking: 10 questions and answers," *IEEE Comm. Magazine*, pp. 18–20, 2017.

[4] M. Selimi, L. Cerdà-Alabern, M. Sánchez-Artigas, F. Freitag, and L. Veiga, "Practical service placement approach for microservices architecture," in *Proceedings of the 17th IEEE/ACM CCGRID*, 2017, pp. 401–410.

[5] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang, and J. Wang, "Large-Scale Measurement and Characterization of Cellular Machine-to-Machine Traffic," *IEEE/ACM Tran. on Networking*, pp. 1960–1973, 2013.

[6] O. Kaiwartya, A. H. Abdullah, Y. Cao, J. Lloret, S. Kumar, R. R. Shah, M. Prasad, and S. Prakash, "Virtualization in wireless sensor networks: fault tolerant embedding for internet of things," *IEEE Internet of Things Journal*, pp. 571–580, 2018.

[7] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for wireless sensor networks," in *Proceedings of IEEE INFOCOM*, 2015, pp. 513–521.

[8] D. Wu, D. I. Arkhipov, E. Asmare, Z. Qin, and J. A. McCann, "UbiFlow: Mobility management in urban-scale software defined IoT," in *proceedings of IEEE INFOCOM*, 2015, pp. 208–216.

[9] M. T. I. ul Huque, W. Si, G. Jourjon, and V. Gramoli, "Large-scale dynamic controller placement," *IEEE Tran. on Network and Service Management*, pp. 63–76, 2017.

[10] L. Ma, S. Yi, and Q. Li, "Efficient service handoff across edge servers via docker container migration," in *Proceedings of the 2nd ACM/IEEE SEC*, 2017, p. 11.

[11] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX Conference on OSDI, 2010*. USENIX Association, 2010, pp. 1–6.

[12] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the 3rd HotSDN, 2014*. ACM, 2014, pp. 1–6.

[13] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proceedings of the 1st Workshop on HotSDN*, 2012, pp. 19–24.

[14] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. R. Kompella, "Elasticon: an elastic distributed sdn controller," in *Proceedings of the 10th ANCS , 2014*. ACM, 2014, pp. 17–28.

[15] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," *SIGCOMM Comput. Commun. Rev.*, pp. 351–362, 2010.

[16] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *SIGCOMM Comput. Commun. Rev.* ACM, 2011, pp. 254–265.

[17] V. Huang, Q. Fu, G. Chen, E. Wen, and J. Hart, "BLAC: A Bindingless Architecture for Distributed SDN Controllers," in *proceedings of 42nd IEEE LCN*, 2017, pp. 146–154.

[18] A. Panda, W. Zheng, X. Hu, A. Krishnamurthy, and S. Shenker, "SCL: simplifying distributed SDN control planes," in *Proceedings of the 14th USENIX Conference on NSDI*. USENIX Association, 2017, pp. 329–345.

[19] "UW CSE | Systems Research | Rocketfuel," May 2005, [accessed 28. Jul. 2018]. [Online]. Available: https://research.cs.washington.edu/networking/rocketfuel

[20] (2018) OpenDaylight. [Online]. Available: http://www.opendaylight.org/

[21] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, pp. 80–83, 1945.

Amalgam: Distributed Network Control With Scalable Service Chaining

Subhrendu Chattopadhyay
IIT Guwahati
Guwahati, India 781039
subhrendu@iitg.ac.in

Sukumar Nandi
IIT Guwahati
Guwahati, India 781039
sukumar@iitg.ac.in

Sandip Chakraborty
IIT Kharagpur
Kharagpur, India 721302
sandipc@cse.iitkgp.ernet.in

Abhinandan S. Prasad
NIE Mysuru
Karnataka, India 570008
abhinandansp@nie.ac.in

***Abstract*—Management of virtual network function (VNF) service chaining for a large scale network spanning across multiple administrative domains is difficult due to the decentralized nature of the underlying system. Existing session-based and software-defined networking (SDN) oriented approaches to manage service function chains (SFCs) fall short to cater to the plug-and-play nature of the constituent devices of a large scale eco-system such as the Internet of Things (IoT). In this paper, we propose *Amalgam*, a composition of a distributed SDN control plane along with a distributed SFC manager, that is capable of managing SFCs dynamically by exploiting the in-network processing platform composed of plug-and-play devices. To ensure the distributed placement of VNFs in the in-network processing platform, we propose a greedy heuristic. Further, to test the performance, we develop a complete container driven emulation framework *MiniDockNet* on top of standard *Mininet* APIs. Our experiments on a large scale realistic topology reveal that *Amalgam* significantly reduces flow-setup time and exhibits better performance in terms of end-to-end delay for short flows.**

***Index Terms*—Service function chaining, Virtual network function, In-network processing, Programmable network, software defined network**

I. Introduction and Related Works

Due to the rapid deployments of connected environments, large-scale Internet of Things (IoT) networks [1] have become prevalent in recent years. Management of such large-scale heterogeneous ecosystems requires various network services such as network address translator (NAT), firewall, proxy, and local domain name server (DNS); these network services are called network function (NF)s. Generally, the network functions are deployed using virtual machines (VM)(s) to provide service isolation and reduce CapEx and OpEx; therefore, they are termed as virtualized network function (VNF) [1]. VNFs execution require computation platform to host the VM and execute the NF within the VM. Depending on network management policies, the application messages require steering through an ordered set of VNFs known as service function chaining (SFC) [2].

Among various existing architectures to execute VNFs over a network infrastructure [3]–[5] relies on software-defined network (SDN) [6] to steer flows from one VNF to another. On the other hand, [7], [8] takes a session-based approach where the end hosts control the SFC. Session-based approaches achieve lower host-based state management of VNFs, where SDN-based approaches achieve fine-grained quality of service (QoS). However, for a large-scale network spanning across multiple administrative domains, both of the SFC management (SCM) approaches fall short in several aspects as follows.

(a) Lack of scalability: Existing SCMs [6], [9] use a central controller that monitors the resource usage of the devices and use as the basis for the VNFs deployment. The use of a central controller for VNF deployment becomes challenging, especially when the network spans across multiple autonomous administrative domains that interconnected through different network service providers. On the other hand, the VNF placement is $\mathcal{NP}$-hard [10]. Existing distributed heuristics for VNF placement [11] require multiple rounds to deploy VNFs, which increases flow initiation delay leading to reduced IoT application performance since the majority of the IoT flows are short-lived [12].

(b) Dynamic service chaining: Usually, VNFss modifying the headers are common in a large-scale network. Consequently, the participating VNFs can change the SFCs during the lifetime of a flow based on the flow characteristics. For instance, a classifier VNF can add a load balancer based on the arrival rate of the packets in a flow. Existing scalable distributed VNF placement methods [11] and IP based traffic steering proposals [13] are not suitable for dynamic service chaining. On the other hand, [7] ascertains dynamic service chaining by adding an agent in each device, including hosts. Installation of agents on a large scale IoT becomes infeasible, where the devices with plug-and-play capability can dynamically enter and exit the ecosystem.

(c) Issues of flow monitoring over multi-administrative platforms: To steer the traffic through proper service chains while ensuring QoS, requires fine-grained flow monitoring. Existing flow identification methods using packet header fields are insufficient in the presence of a header modifying VNF in the SFC (such as NAT, load balancer, and proxy). Existing SDN-based flow monitoring schemes like FlowTags [9], Stratos [14] utilize *"vlan/mpls"* tagging which does not work through multiple administrative domains. On the other hand, the use of packet encapsulation in session-based approaches

This work was supported by TCS India Pvt. Ltd.

[1]https://www.sigfoxcanada.com/

Annex to ISBN 978-3-903176-28-7 © 2020 IFIP

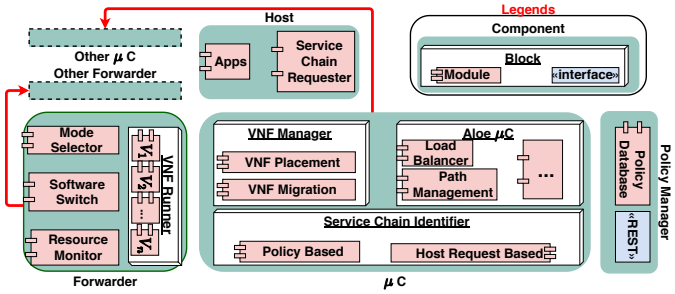


Fig. 1: Component diagram of Amalgam

[8] also fails to ensure QoS.

Therefore, in this paper, we propose *Amalgam* to provide a scalable SFC orchestration platform to provide fine-grained QoS over multiple administrative domains and dynamic SFC. The SCM *Amalgam* provides significant performance improvement in terms of flow initiation delay. The step by step contributions is as follows.

- *Amalgam* achieves distributed state management while ensuring fine-grained QoS by coupling distributed SDN and distributed SCM.
- We develop a distributed heuristic for the distributed deployment of VNFs, which provides performance improvement in terms of flow initiation delay.
- *Amalgam* design exploits the "micro-service"(μS) architecture of the in-network processing platform [15] to support dynamic service chaining and "`zero-touch deployment`" to support the plug and play nature of the devices.
- For performance evaluation, we develop an emulation framework *MiniDockNet* for VNF deployment using "`docker`" [2] over in-network processing, as the existing network name-space oriented mininet [3] emulator is not sufficient for in-network processing.
- Based on the experimental results over a realistic large-scale system (with 70 devices and 6 different service chain scenarios), we found that *Amalgam* can ensure fine-grained QoS without significant increase of resource utilization of the devices. After comparing the performance of *Amalgam* with two state-of-the-art distributed SFC platform *"Dysco"* [7] and *"WGT"* [11] we found that, *Amalgam* is capable of a significant reduction in the flow initiation delay and improves the performances of short-duration flows in terms of end-to-end delay.

This paper is organized as follows. Section II describes the architecture of proposed *Amalgam*. Section III describes the design choices taken during the development of *Amalgam*. We describe the experimental setup details, results and our observations in Section IV before concluding in Section V.

II. Architecture

The proposed *Amalgam*[4] is constructed on the top of *Aloe* [16] framework. *Aloe* provides an orchestration framework for a fault-tolerant self-stabilizing distributed control plane on top of the in-network processing platform using μCs (microcontrollers) instead of the standard SDN controller. Like *Aloe*, *Amalgam* supports 3 operating modes for each device; (a) **Host/(default) mode**: if the device executes only the client/server application, (b) **Forwarding mode**: If the device has multiple active network interfaces, (c) **μC mode**: based on the topology, if *Aloe* selects the device as a μC. At any instant, each device is in "`atleast`" any one of the above modes and each mode is a component of *Amalgam* framework as shown in Fig. 1. A mode selector module in *"forwarder"* component identifies the mode of operation and activates the respective components. Apart from the mode functional components, we add a seperate *"policy manager"* component in the *Amalgam* framework. Policy manager is a distributed database with a *"REST"* driven interface which contains (i) the flow identifier (i.e., "`OpenFlow`" match field) and (ii) ordered list of the types of VNF (service chain) through which the flow should be steered. This component is consulted by the mode functional components at time of determination of SFC.

Among the mode functional components, the host is the simplest. This component is responsible for traffic generation through the *"App"* module, which represents the client/server applications. Additionally, this module can request the nearest μC to change the SFC for the flows generated by the host. The *"mode selector"* module elevates the mode of the device with multiple active interfaces to *"forwarder"*. The forwarder component consists of *"software switch"*, *"VNF runner"*, and a *"resource monitor"*. The software switch module is responsible for forwarding data from one interface to another. On the other hand, the *"VNF runner"* block is reserved for execution of VNFs (e.g., V_1, V_2 etc. as shown in Fig. 1). This VNF runner block from each device constructs the in-network processing framework. During the execution of the VNFs *"resource monitor"* module periodically monitors the available resources in the device. The resource monitor module forwards the collected resource utilization statistics to the μC component.

The μC component is composed of three functional blocks namely Service chain identifier (SCI), ""*VNF manager*"", and "*Aloe* μC". The tasks of these blocks are as follows.

1) Service Chain Identifier: At the startup phase of the μC, SCI caches the policy in a local cache. The local cache is updated whenever the policy manager database is updated. SCI module is consulted when an "`OpenFlow`" "`packet in`" event is initiated at the μC. From the list of VNFs in the service chain, SCI chooses the first VNF, and it's execution status in the local domain. If the VNF is executing inside a forwarder connected to the μC, the SCI consults the path management module to establish the data flow path by installing flow table entries via standard "`OpenFlow`" protocol. Otherwise, it sends a search query to the other μCs to identify the target VNF address. If the address of the VNF is not found, then SCI consults the VNF manager module (Section II-2) to start the execution of the VNF. This procedure is iterated for all the VNFs in the service chain.

[2]https://www.docker.com/
[3]http://mininet.org/
[4]https://github.com/subhrendu1987/NFV_MiniDockNet

2) VNF Manager: The VNF manager module (VMM) works in a distributed fashion and communicates with the neighbor μCs. VMM tries to answers the following two questions; (a) should the VNF be placed in any of the forwarders associated with the μC? (b) which forwarder should take care of the VNF?. The detailed protocol to find an answer to these questions is described in next section. Additionally, VMM also takes care of the dynamic addition or removal of the VNFs to an ongoing flow.

3) Aloe μC: The *Aloe* μC module is the containerized SDN μ controller module as described in [16]. *"Aloe"* provides a distributed fault-tolerant controller module suitable for in-network processing frameworks that are responsible for path management. Use of *Aloe* ensures quick flow initiation along with fault and partition tolerance in *Amalgam*. However, during the design of *Amalgam*, we face several challenges exclusive to SFC deployment of IoT

III. Challenges and Design Choices

The goal of *Amalgam* is to provide a highly dynamic in-network processing platform. In this section, we describe the implementation challenges and the proposed solutions to overcome the scalability issues without affecting the dynamic behavior of the platform.

(a) Plug-and-Play Capability:A typical IoT platform is composed of plug-and-play devices where "`zero-touch deployment`" [5] is highly desired. It is necessary to configure a new device as soon as it enters the eco-system. To avoid individually configuring the devices, we design each component of *Amalgam* (except the host component) as Docker containers. Once a device enters the eco-system, it assumes the host mode of operation. Since the host mode does not require anything more than the IoT applications (clients and servers), they can work smoothly. Whenever the device wishes to change its mode, it can pull the container image of the *Amalgam* component from the nearest forwarder.

(b)Distributed VNF Placement: In a short-lived flow heavy system, minimization of the flow initiation delay is critical. The flow initiation delay consists of following components namely (a) Controller consultation delay (b) SFC deployment delay, and (c) path setup delay. The proposed VNF placement reduces the SFC deployment delay. A SFC for a particular flow is composed of multiple VNFs, which requires resource consumption. Each device of a IoT in-network processing platform has residual resources that can be used for deployment of these VNFs's. The proposed VNF placement identifies the set of devices where the VNFs of the SFCs can be placed for a given network and flow profile while satisfying the capacity constraints of the devices. Maintaining capacity constraints in a multi-domain system is non-trivial since the residual capacity of a device residing in a different administrative domain is difficult to collect. Therefore, we propose the greedy heuristic as given in the Algorithm 1.

[5]https://www.etsi.org/technologies/zero-touch-network-service-management

Algorithm 1: Distributed Placement of VNF

Function `GreedyPlace` (*Path:* P^a, *Service Chain:* C^j, μC: l):
  Find ordered set of unplaced VNFs from C^j;
  $I \leftarrow \{i : i \in P^a, \varphi_i = l\}$;
  Place as many VNFs as possible among I;
  return number of VNFs placed;

Function `Main` (*Flow:* f^j,μC: l):
  /* Find VNF placement profile for f^j in φ_i */
  Find set of paths (P) from s_j to d_j by querying "Path Management" module of l;
  $\underset{P^a \in P}{maximize}$ `GreedyPlace` (P^a, C^j, l);
  if $\exists c_{j,k} \cdots c_{j,j_{max}}$ *not placed* **then**
    /* All devices under l */
    Obtain the list of adjacent μC of l and store it in $N\mu$**foreach** $l' \in N\mu$ **do**
      `Main` (f^j, l');
  return;

Each μC in the end-to-end path (P) executes the proposed heuristic for each flow (f^j represents jth flow) from source (s_j) to destination (d_j). We denote SFC of f^j with C^j. Certain μC with ID l maintains the topology information as the list of devices (D_l) and list of links (E_l) where each link $e_{i,i'} \in E_l$ represents the physical connection between two devices (i and i'). For the sake of simplicity, we denote the μC associated with $i$as φ_i. The proposed heuristic identifies a path P^a between s_j to d_j from the set of P such that, most of the VNFs of C^j are placed near s_j in a distributed fashion. This way, one μC does not need the resource utilization of devices from other administrative domains. Once the flow is established, the resource utilization of devices in the path (`info`) is piggybacked with the data packets. The VNF manager can re-solve the Algorithm 1 and find a new allocation of VNF with updated utilization.

(c)Migrations of the VNFs: A VNF may be relocated during (a) VNF readjustment due to prior sub-optimal placement and (b) addition or removal of the device. The μC nearest to the source node of the flow decides the VNF readjustment after it receives the piggybacked resource utilization of the devices. On the other hand, the addition and removal of devices trigger "`topology_change`" event, and the local μC initiates the decision about the VNF deployment. In both cases, the decider μC starts the migration process at the source device. Initially, the source device saves a snapshot of the executing container, and the snapshot is transferred and restored in the target device. Finally, the μC updates the existing flow table entries accordingly.

(d)Dynamic management of service chains: *Amalgam* provides support for dynamic service chaining. Dynamic service chaining enables VNFs to meet changing service requirements. For instance, consider a flow passes through a firewall VNF. Based on the signature of the flow, the firewall conditionally decides to steer the flow through an additional deep packet inspector (DPI) without interrupting the flow. To implement this, *Amalgam* allows the VNFs to interact with the local μC via "`REST`" interface. The local μC can deploy the DPI if it is not available and sends the "`OFPT_FLOW_MOD`" events to

the forwarder component to enable the flow steering without terminating the flow.

(e)Flow Tags for Monitoring: Once the VNFs are placed, the path management module of μCs set-up the flow table entries of the participating forwarders via `OpenFlow` protocol. One issue regarding path management through service chains is to identify an end-to-end flow that arises in the presence of the "5-tuple" changing VNFs (e.g., Load balancer, web proxy cache, and NATs). Since such VNFs may alter the packets in unpredictable ways, fine-grained management and monitoring of the flows passing through becomes difficult. To avoid this issue, *Amalgam* attaches a "`VLAN`" tag to the packets before it enters the VNF. The nearest μC of the VNF maintains a table for flows like f^a, which keeps track of the original match field of the flow and the modified field (alias).

(f)Providing QoS: *Amalgam* is developed on top of the SDN decentralized control plane, which enables us to ensure flow specific QoS guarantees. On the other hand, since the VNF deployment is done using containers, using "`cgroups`"can ensure the VNF specific QoS like reservation of CPU, Memory, etc. The policy server module contains the "`cgroups`" parameters for each VNFs of a service chain, which is used to ensure VNF specific QoS.

IV. Prototype and Experimental Results

The existing namespace oriented emulation frameworks (e.g. Mininet) is not suitable for VNF migration and in-networking platform emulation. Therefore, we develop docker based MiniDockNet which mimics real-life VNFs using the *"Docker-in-Docker"* configuration. This feature ensures rapid deployment from MiniDockNet emulation to real in-network processing environment. To implement the VNF migration, we use standard live container migration using CRIU[6]. For the emulation of the links between any two nodes, we use *"l2tp"*

A. Experimental Setup

For experimental purpose, we use "`rocketfuel topology`"[7]. Each link is configured to emulate $3ms$ of delay and $10Mbps$ of bandwidth using linux "`tc`" utility. We use "`iperf`" to generate long flows; for shorter flows we use "`ping`". The clients and server applications are hosted on the *diameter* of the topology. For background traffic we use python based "`HTTP`" client and server.

We use "`Apache cassendra`" to implement the policy server module. Rest of the *Amalgam* modules targeted for μC are implemented on top of "`Ryu`"[8], a python based SDN controller framework. For experiments, we use 3 different VNFs (NAT (N), Load Balancer (L) and Web Proxy(W)) to create 6 different combination of service chain as given in Fig. 5. In order to ensure the confidence on the results, each experiment is repeated atleast 30 times.

[6]https://criu.org/Live_migration

[7]http://tiny.cc/nv70mz

[8]https://ryu.readthedocs.io/en/latest/

B. Results

We compare the performance of *Amalgam* with the existing *"P4"*[9] based distributed session-oriented service function chaining framework called Dysco [7]. Since, Dysco ensures session related performance and does not provide any VNF placement strategy, the performance evaluation of the proposed distributed VNF placement algorithm is done with another existing work WGT [11] which proposes a distributed heuristic for VNF placement for the multi-domain network.

1) Session Related Performances: Fig. 2 shows the comparison between Dysco and *Amalgam* in terms of flow initialization delay. We found that *Amalgam* is capable of quicker flow initialization than Dysco. This reduction in flow initialization delay comes from the parallel deployment of VNFs as opposed to the hop by hop deployment of VNFs in Dysco. The advantage of flow initialization delay becomes much evident in the case of longer service chains like C^6 than the smaller service chain like C^1.

Since *Amalgam* uses containers to deploy the VNFs as opposed to the P4 applications used in Dysco, the deployment of VNFs using *Amalgam* incurs greater latency, as shown in Fig. 3. The increase in VNF deployment time for *Amalgam* depends on the VNF container size. Therefore, the deployment latency is higher for C^6 in compared to C^1. However, in a large scale network, VNF deployment events are far rare than a flow generation event. On the other hand, the use of containers provide greater flexibility as the creation of new middlebox application using container requires less programming overhead than the creation of a new P4 application. As a result, state management during the migration of VNFs from one node to another becomes easy when they are running inside a container as compared to the P4 applications of Dysco. However, these management benefits of containers come at the cost of resource utilization.

The placement of VNFs requires resource occupancy in the deployed devices, which is an important aspect of resource constraint IoT devices. In Fig. 6, we compare the performance of *Amalgam* with Dysco in terms of CPU utilization of devices due to the placement of VNFs. In order to normalize the additional resource consumption of *Amalgam* due to the use of containers, we also compare the resource utilization of *Amalgam* without using docker. Similarly, we provide a comparison of memory utilization for *Amalgam* and Dysco in Fig. 7. Based on these two experiments, we observe that Dysco incurs less utilization of resources than the proposed *Amalgam* with the container. However, based on the *"Wilcoxon Rank Sum test"* we find that, the difference of resource utilization of *Amalgam* without Docker and Dysco is statistically insignificant (i.e. $p-value > 0.05$) for C^4, C^5 and C^6. Fig. 8 shows the comparison of throughput between *Amalgam* and Dysco. The Wilcoxson rank sum test reveals that the throughput between *Amalgam* and Dysco are statistically indistinguishable (Here our alternate hypothesis H_a is *Amalgam* provides less throughput than Dysco).

[9]https://p4.org/

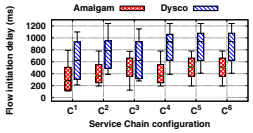


Fig. 2: Flow Initialization Delay

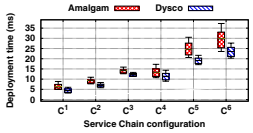


Fig. 3: Latency of Deployment

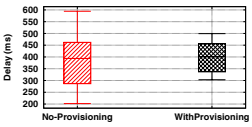


Fig. 4: Effect of QoS

Name	SFC	Name	SFC
C^1	(N)	C^2	(L)
C^3	(W)	C^4	(N→L)
C^5	(L→W)	C^6	(N→L→W)

Fig. 5: List of Service Chains

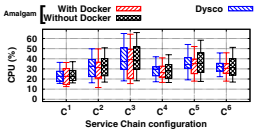


Fig. 6: CPU Utilization

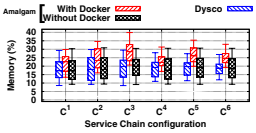


Fig. 7: Memory Utilization

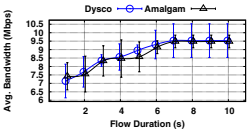


Fig. 8: Average Throughput

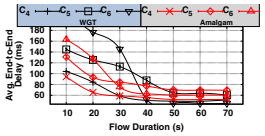


Fig. 9: Delay for C^4, C^5 and C^6

2) Performance of Distributed VNF Placement: To measure the performance of distributed VNF placement heuristic used in Amalgam, as mentioned earlier, we deploy WGT [11] on top of Dysco. However, it is difficult to deploy a centralized controller for a large scale multi-domain system. Therefore, we place the WGT in the micro-Controller (μC) nearest to the source device. We measure and compare the effect of delay for all the service chains when the flow duration increases. Based on the experimental results, we found that the effect of delay for single VNF does not change since *Amalgam* and WGT provides the same results for VNF placement. Hence, we omit the plots for C^1, C^2, C^3. For multiple VNF oriented service chains like C^4, C^5 and C^6, we provide the average end-to-end delay in Fig. 9. Based on the results, we can observe that *Amalgam* can perform significantly well for shorter flows as the iterative WGT requires a significant amount of feedback rounds to find the proper placements of VNFs.

C. QoS Provisioning

Amalgam is capable of showing QoS provisioning by reserving resources limiting CPU, memory, bandwidth, and link delay. We perform two experiments for each resource type, one with no provisioning and another with resource reservation limit set as the mean value found in the previous experiment. Based on the Wilcoxson rank-sum test on these results we found that, except the memory utilization (P-value $= 0.42$) rest of the resource reservation works significantly well (with P-value< 0.05). We also find that the resource reservation can reduce the jitter of the flow, as shown in Fig. 4.

V. Conclusion

In this paper, we present *Amalgam*, which integrates the distributed SDN orchestration framework with the distributed service chain management framework. The proposed *Amalgam* is suitable for large scale multi-domain IoT in-networking platforms. We also provide a distributed heuristics for the placement of constituent VNFs of service chains. The lack of an existing emulation platform for container oriented VNF service chain has motivated us to develop "MiniDockNet". Using this emulation platform, we found that *Amalgam* incurs a lesser flow initialization delay than that of a very recent distributed service chain management framework (Dysco). We also show that *Amalgam* is capable of ensuring less end-to-end delay for short flows.

References

[1] F. Duchene, D. Lebrun, and O. Bonaventure, "Srv6pipes: enabling in-network bytestream functions," in *17th IFIP Networking Conference and Workshops*, May 2018, pp. 1–9.

[2] W. Haeffner, J. Napper, M. Stiemerling, D. Lopez, and J. Uttaro, "Service function chaining use cases in mobile networks," *IETF*, 2015.

[3] D. Bhamare, R. Jain, M. Samaka, and A. Erbad, "A survey on service function chaining," *J. Netw. Comput. Appl.*, vol. 75, no. C, pp. 138–155, nov 2016.

[4] S. Rajagopalan, D. Williams, H. Jamjoom, and A. Warfield, "Split/merge: System support for elastic execution in virtual middleboxes," in *10th USENIX Symposium on NSDI*. Lombard, IL: USENIX, 2013, pp. 227–240.

[5] A. Gember-Jacobson, R. Viswanathan, C. Prakash, R. Grandl, J. Khalid, S. Das, and A. Akella, "OpenNF: Enabling innovation in network function control," *SIGCOMM Computer Communication Review*, vol. 44, no. 4, pp. 163–174, 2015.

[6] A. Mohammadkhan, S. Ghapani, G. Liu, W. Zhang, K. K. Ramakrishnan, and T. Wood, "Virtual function placement and traffic steering in flexible and dynamic software defined networks," in *The 21st IEEE International Workshop on LAN/MAN*, April 2015, pp. 1–6.

[7] P. Zave, R. A. Ferreira, X. K. Zou, M. Morimoto, and J. Rexford, "Dynamic service chaining with dysco," in *30th Proceedings of the ACM SIGCOMM*. ACM, 2017, pp. 57–70.

[8] IETF, "Rfc 8300 - network service header (NSH)," Sept 2019, [accessed 10. Sept. 2019]. [Online]. Available: https://tools.ietf.org/html/rfc8300

[9] S. K. Fayazbakhsh, L. Chiang, V. Sekar, M. Yu, and J. C. Mogul, "Enforcing network-wide policies in the presence of dynamic middlebox actions using flowtags," in *11th USENIX Symposium on NSDI*. Seattle, WA: USENIX Association, 2014, pp. 543–546.

[10] M. Ghaznavi, N. Shahriar, S. Kamali, R. Ahmed, and R. Boutaba, "Distributed service function chaining," *IEEE Journal on Selected Areas in Communications*, vol. 35, no. 11, pp. 2479–2489, Nov 2017.

[11] G. Sun, Y. Li, D. Liao, and V. Chang, "Service function chain orchestration across multiple domains: A full mesh aggregation approach," *IEEE Transactions on Network and Service Management*, vol. 15, no. 3, pp. 1175–1191, Sep. 2018.

[12] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang, and J. Wang, "Large-scale measurement and characterization of cellular machine-to-machine traffic," *IEEE/ACM Transactions on Networking*, vol. 21, no. 6, pp. 1960–1973, Dec 2013.

[13] A. Wion, M. Bouet, L. Iannone, and V. Conan, "Let there be chaining: How to augment your igp to chain your services," in *18th IFIP Networking Conference*, May 2019, pp. 1–9.

[14] A. Gember, A. Krishnamurthy, S. S. John, R. Grandl, X. Gao, A. Anand, T. Benson, A. Akella, and V. Sekar, "Stratos: A network-aware orchestration layer for middleboxes in the cloud," *arXiv CoRR*, vol. abs/1305.0209, 2013.

[15] M. Charikar, Y. Naamad, J. Rexford, and X. K. Zou, "Multi-commodity flow with in-network processing," in *Algorithmic Aspects of Cloud Computing*. Cham: Springer International Publishing, 2019, pp. 73–101.

[16] S. Chattopadhyay, S. Chatterjee, S. Nandi, and S. Chakraborty, "Aloe: An elastic auto-scaled and self-stabilized orchestration framework for iot applications," in *37th IEEE Proceedings of INFOCOM*. Paris, France: IEEE, 2019, pp. 802–810.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.3008426, IEEE Transactions on Network and Service Management

Aloe: Fault-Tolerant Network Management and Orchestration Framework for IoT Applications

Subhrendu Chattopadhyay , Soumyajit Chatterjee , Sukumar Nandi , Sandip Chakraborty

***Abstract*—Internet of Things (IoT) platforms use a large number of low-cost resource constrained devices and generates millions of short-flows. In-network processing is gaining popularity day by day to handle IoT applications and services. However, traditional software-defined networking (SDN) based management systems are not suitable to handle the plug and play nature of such systems. In this paper, we propose Aloe, an auto-scalable SDN orchestration framework. Aloe exploits in-network processing framework by using multiple lightweight controller instances in place of service grade SDN controller applications. The proposed framework ensures the availability and significant reduction in flow-setup delay by deploying instances in the vicinity the resource constraint IoT devices dynamically. Aloe supports fault-tolerance with recovery from network partitioning by employing self-stabilizing placement of migration capable controller instances. Aloe also provides resource reservation for micro-controllers so that they can ensure the quality of services (QoS). The performance of the proposed system is measured by using an in-house testbed along with a large scale deployment in Amazon web services (AWS) cloud platform. The experimental results from these two testbeds show significant improvement in response time for standard IoT based services. This improvement of performance is due to the reduction in flow-setup time. We found that Aloe can improve flow-setup time by around $10\% - 30\%$ in comparison to one of the states of the art orchestration framework.**

***Index Terms*—Orchestration Framework, Fault-tolerance, Programmable network, IoT, In-network processing, SDN**

I. Introduction

The rapid proliferation and deployments of large-scale Internet-of-Things (IoT) applications have made the network architecture complicated from the perspective of end-user service management. Simultaneously, with the advancement of edge-computing, in-network processing and platform-as-a-service technologies, end-users consider the network as a platform for the deployment and execution of myriads of diverse applications dynamically and seamlessly. Consequently, network management has become increasingly difficult in today's world with this complex service-oriented platform overlay on top of the inherently distributed TCP/IP network architecture. The concept of software-defined networking (SDN) [1] has gained popularity over the last decade to make the network management simple, cost-effective and logically centralized, where a network manager can monitor, control and deploy new network services through a central controller. Nevertheless, edge and in-network processing over an IoT platform is still challenging even with an SDN based architecture [2].

Motivation: The primary motivation for supporting a distributed network management and flow steering framework coupled with edge and in-network processing over an IoT platform are as follows: (1) The platform should be agile to support rapid deployment of applications without incurring additional overhead, ensuring the scalability of the system [3], [4]. (2) With micro-service architectures [5], in-network processing requires dividing a service into multiple micro-services and deploying them at different network nodes for reducing the application response time with parallel computations. However, such micro-services may need to communicate with each other, and therefore the flow-setup delay from the in-network nodes need to be very low to ensure near real-time processing. (3) The percentage of short-lived flows are high for IoT based networks [6]. This also escalates the requirement for reducing flow-setup delay in the network. (4) Failure rates of IoT nodes are in-general high [7]. Therefore, the system should support a fault-tolerant or fault-resilient architecture to ensure liveness.

Limitations of existing approaches: Although SDN supported edge computing and in-network processing have been widely studied in the literature for the last few years [2], [8], [9] as a promising technology to solve many of the network management problems, they have certain limitations. First, the SDN controller is a single-point bottleneck. Every flow initiation requires communication between switches and the controller; therefore, the performance depends on the switch-controller delay. With a single controller bottleneck, the delay between a switch and the controller increases, which affects the flow-setup performance. As we mentioned earlier, the majority of the flows in an IoT network are short-lived, the impact of switch-controller delay is more severe on the performance of short-lived flows. To solve this issue, researchers have explored distributed SDN architecture with multiple controllers deployed over the network [10]. However, with a distributed SDN architecture, the question arises about how many controllers to deploy and where to deploy those controllers. Static controller deployments may not alleviate this problem, as IoT networks are mostly dynamic with plug-and-play deployments of devices. Dynamic controller deployment requires hosting the controller software over commercially-off-the-shelf (COTS) devices and designing methodologies for controller coordination, which is a challenging task [11]. The problem is escalated with the objective of developing a fault-tolerant or fault-resilient architecture in a network where the majority of the flows are short-lived flows.

S. Chattopadhyay and S. Nandi are with the Department of CSE, Indian Institute of Technology Guwahati, India 781039. email:{subhrendu, sukumar}@iitg.ac.in. S. Chatterjee and S. Chakraborty are with Department of CSE, Indian Institute of Technology Kharagpur, India, 721302. email: {soumyachat@iitkgp.ac.in, sandipc@cse.iitkgp.ac.in}

Our contribution: To alleviate the above mentioned challenges, we, in this paper, integrate a distributed SDN control plane with the in-network processing infrastructure, such that the control plane can dynamically be deployed over the COTS devices maintaining a fault-tolerant architecture. We design a distributed, robust, migration-capable and elastically scalable control plane framework with the help of docker containers [12] and state-of-the-art control plane technologies. The proposed control plane consists of a set of small controllers, called the *micro-controllers*, which can coordinate with each other and help in deploying new applications for in-network processing. The container platform helps in installing these micro-controllers on the COTS devices; a container with a micro-controller can be seamlessly migrated to another target device if the host device fails, yielding a fault-tolerant architecture. In addition to this, the deployment mechanism for the micro-controllers ensure elastic auto-scaling of the system; the total number of controllers can grow or shrink based on the number of active devices in the IoT network. We develop a set of special purpose programming interfaces to ensure fault-tolerant elastic auto-scaling of the system along with intra-controller coordination. Finally, we design a set of application programming interfaces (API) over this platform to ensure language-free independent deployment of applications for in-network processing. Combining all these concepts, we present Aloe, a distributed, robust, auto-scalable, platform-independent orchestration framework for edge and in-network processing over IoT infrastructures.

Aloe has multiple advantages for a IoT framework with in-network processing capabilities. (a) The distributed controller approach ensures that there is no performance bottleneck near the controller. (b) The flow-setup delay is significantly minimized because of the availability of a controller near every device. (c) The fault-tolerant controller orchestration ensures the liveness of the system even in the presence of multiple simultaneous devices or network faults. An initial version of Aloe has been presented in IEEE INFOCOM 2019 [13]. The current version of the paper gives multiple additional features of Aloe, along with formal proofs of its characteristics, such as the convergence, the correctness and the closure of the self-stabilization algorithm used in Aloe micro-controller deployment. We additionally discuss various trade-offs for Aloe deployment and service provisioning performance, based on thorough experimentation and performance analysis under various realistic scenarios. Accordingly, we introduce a resource management module to Aloe, which boost up the performance under dynamic workload scenarios.

We have implemented a prototype of Aloe using state-of-the-art SDN control plane technologies and deployed the system over an in-house testbed and a 68-node Amazon web services platform. The in-house testbed consists of 10 nodes (Raspberry Pi devices) with Raspbian kernel version 8.0. As mentioned, we have utilized docker containers to host the distributed control plane platform. We have tested Aloe with three popular applications for in-network IoT data processing – (a) A web server (simple python based), (b) a distributed database server (`Cassandra`), and (c) a distributed file storage platform (`Gluster`). We observe that Aloe can reduce the flow-setup delay significantly (more than three times) compared to state-of-the-art distributed control plane technologies while boosting up application performance even in the presence of multiple simultaneous faults.

The rest of the paper is organized as follows. § II discusses about the existing works related to our problem. The basic design and various components of Aloe are described in § III. § IV and § V discuss the design and implementation of the proposed Aloe orchestration framework. In § VI, we present the evaluation results from both in-house testbed and large-scale AWS cloud deployments. § VII discusses about various extensions of Aloe exclusive to this paper, where we provide a time series based prediction method for resource reservation of Aloe micro-controllers to ensure the quality of services (QoS). Finally we conclude the paper in § VIII.

II. Related Work

SDN distributed control plane [14]–[17] provides scalability which is necessary for practical large scale IoT systems. ONIX [18] and ONOS [19] are two popular distributed control plane architectures. ONIX uses a distributed hash table (DHT) data store for storing volatile link state information. On the other hand, ONOS uses NoSQL distributed database and distributed registry to ensure data consistency. Although both of them can scale easily and show a significant amount of fault-resiliency, they require high end distributed computing infrastructure for execution. Deployment of such infrastructure increases the cost of IoT deployment and leads to performance degradation of IoT services. To tackle the high resource requirements, Elasticon [20] uses controller resource pool to enforce load balancing. They also proposed a hand-off protocol for switch controller co-ordination to ensure the serializability. However, Elasticon is not suitable for failure prone IoT nodes. Similar problem is also faced by Kandoo [21] and [22]. A Markov chain based formal model has been proposed in [23] to analyse the dependability of optical networks over double ring topology. However, maintaining a double ring topology in a dynamic IoT system is challenging. The fault-tolerant SDN control plane is discussed in [24]–[26], where they have used multiple replica controllers, which can take the place of a failed controller. However, the use of replica controllers provides fault-resilience based on the number of replica controllers. On the other hand, managing state consistency between the primary controllers and the slave controllers is difficult in an IoT network with a large amount of flows. To overcome these issues, Aloe proposes an auto-scalable, distributed, and fault-tolerant SDN control plane using "self-stabilizing" controller deployment.

IoT applications generate short and bursty traffics. To avoid the impacts of short flows, DIFANE [28] uses special purpose authority switches. The authority switches can take localized decisions based on pre-installed wildcard flow entries depending on the traffic characteristics and network topology. However, local authority switches create a problem for the global state management of the network. DevoFlow [29] tries to solve this problem by proactively deciding wild-carded rules based on global state information. However, the dynamic

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.3008426, IEEE Transactions on Network and Service Management

TABLE I: Comparison of Existing Works

Names	Key contributions	Issues			
		Plug and Play	In-band control	Short flow friendly	Arbitrary failure
ONIX [18], ONOS [19]	Distributed network state	N	Y	N	N
Elasticon [20]	Controller load balancing	N	N	Y	N
Kandoo [21]	Hierarchical control plane	N	N	N	Y
BLAC [27]	Controller scheduling	N	N	Y	N
DIFANE [28], DevoFlow [29]	Pro-active flow installation	N	Y	N	N
SCL [30], ASCA [26]	Replication based In-band control	N	Y	Y	N
RAMA [25]	Consistency preservation	N	N	N	Y

TABLE II: Selected Abbreviations Used

Abbreviation	Explanation
SDC	Service deployment controller
SNC	Super network controller
P2NM	PushToNode Module
TMM	Topology Management Module
SDM	State Discovery Module
RMM	Resource Management Module
MIS	Maximal independent set
DHT	Distributed hash table
COTS	Commercially off the shelf

topology of the IoT platform prevents the proactive installation of flow entries. Therefore, although DIFANE and DevoFlow perform well in the case of a data center network, it delivers substandard performance in the case of IoT platforms.

IoT requires in-band control plane as most of the switches have limited network interfaces. Therefore, disruption in IoT links impacts severely on multiple IoT nodes due to disconnection from the control plane. To provide disruption tolerance, SCL [30] uses replication of controller applications on strategic places of a network. SCL uses a coordination layer inside the switch to provide consistent updates for a single image, lightweight controllers deployed in an in-band fashion. However, the use of a two-phase commit mechanism for consistency preservation increases higher latency and affects the flow setup delay for the short flows. Moreover, SCL assumes the existence of robust channels among switches and controllers, which is not possible in the case of low-cost and resource-constrained IoT platforms. On the other hand, DIFANE, DevoFlow and SCL exploit the data plane device capabilities to provide quicker response time. In order to do so, they require special purpose switches that can take decisions locally without requiring controller consultation. Such switch-level modifications may not be possible for every hardware device. To avoid such scenarios, the proposed Aloe exploits in-network processing platforms of IoT to deploy lightweight controller instances to realize "control plane as service".

To avoid hardware modification, BLAC [27] uses a controller scheduling mechanism to dynamically scale the control plane to accommodate the need of the system. BLAC introduces a scheduling layer to achieve binding less architecture, where all the flows from a switch can be dynamically scheduled to one of the many controller instances. Although, BLAC reduces the switch hand-off issues, increases the flow setup time. Therefore, BLAC re-introduces performance bottleneck for IoT short flows.

From the discussion above and from the comparison of existing works given in Table I we can observe that the states of the art control plane architectures are not effective in managing IoT platforms. The reason behind the non-compatibility is the existing works pour more focus on consistency of the network state information than the availability and partition tolerance of the control plane. However, theoretically it is not possible to achieve all these goals simultaneously [31]. However, we feel that availability and partition tolerance requires more attention than providing strong consistency for IoT platforms. The reason behind this is a dynamic and failure prone network like IoT requires a highly available control plane than preserving consistency for volatile short flow information. The proposed Aloe ensures minimal flow initiation delay by using a self-stabilizing linear time convergent lightweight controller placement adjacent to the switches.

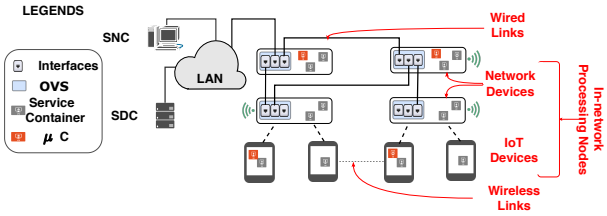


Fig. 1: Components of Aloe Infrastructure

III. Components of Aloe

The Aloe orchestration framework exploits the capabilities of the in-network processing architecture over an IoT platform where devices work mostly in a plug-and-play mode. The main components of the architecture are shown in Fig. 1. It can be noted here that the proposed architecture does not bring new hardware or software platforms at its base; instead, we utilize the available COTS hardware and open-source software suites to design this entire architecture. Our objective is to design an orchestration platform that can be developed with market-available components while integrating innovations in the design such that the shortcomings of the existing systems can be mitigated. We discuss the individual components and their functionalities in this section.

A. Infrastructure Nodes

The networking equipment and devices are considered as the infrastructure nodes. Therefore, nodes are essentially embedded and resource-constraint devices like smart-gateways, smart routers, smart IoT monitoring devices, etc. These devices participate in communication and provide in-network processing platforms for lightweight services by utilizing residual resources. We consider that these nodes are either SDN-supported or can be configured with open-source software platforms like *open virtual switch* (OVS) to make them SDN capable.

We use containerized platforms like docker [12] to offload services in the IoT platform for in-network processing. The containerized service deployment helps in supporting service isolation and makes the architecture failsafe by supporting live migration of containers. Further, containers reduce a

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.3008426, IEEE Transactions on Network and Service Management

programmer's overhead for service delegation and cost of deployment, as the same device can be used for in-network processing of IoT applications along with the execution of custom networking services.

B. Service Deployment Controller (SDC)

To identify the resource requirement and delegation of the services which require in-network processing, we use a centralized service deployment controller (SDC). The SDC periodically monitors the resource consumptions of the nodes. Once a new service is ready for deployment in the system, SDC identifies the schedules in which the nodes can execute the services without violating resource demands from individual services. Once the schedule is generated, the SDC is responsible for delegating the services based on the schedule. It can be noted that the load of an SDC is much less compared to the network management controller. Therefore we maintain a single instance of SDC in our system.

C. Super Network Controller (SNC)

Network management in an IoT platform is non-trivial due to the diversified inter-service communication requirements and the dynamic nature of the network. Aloe uses a two-layer approach. We deploy a high availability super network controller (SNC) [1] at the first layer, which is responsible for storing persistent network information, like routing protocols, QoS requirements, the periodicity of statistic collection from nodes, etc. A SNC also manages an access control list (ACL) to provide necessary security to the infrastructure nodes.

D. Micro-Controller (μC

Although SDC(s) and SNC(s) are highly available, an IoT platform has a time-varying topology due to the use of the resource constraint devices and the devices being plug-and-play most of the time. Therefore, the use of a centralized controller cannot achieve fault-tolerance (failure of infrastructure nodes) and partition-tolerance (failure of network links resulting in network partitions). On the other hand, unlike SDC, SNC needs to be consulted by the nodes each time a new flow enters the system. This consultation overhead increases the communication overhead and flow initiation delay, which also affects the performance of the services deployed in the infrastructure. Therefore, Aloe uses a second layer of network controllers named as *"micro-controllers"* (μC).

μCs are lightweight SDN controllers. Each μC stores volatile link layer information of a small group of nodes placed topologically close to it. Thus a μC maintains information consistency by minimizing the delay between the governing μC and the nodes managed by it. The SNC can aggregate these statistics via `REST API` queries from the μC. Based on the changing QoS of the services, network service provisioning can be achieved in the μC via the same `REST API`. Based on the configuration of the SNC, a μC collects statistics from individual OVS modules of the nodes. Thus a μC can achieve a fine-tuned network control for the infrastructure nodes.

[1]It is possible to use same physical device as SDC and SNC

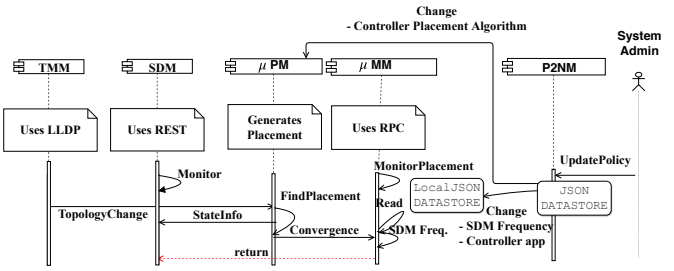


Fig. 2: Aloe function modules and their interactions

However, deployment of μCs in nodes might also create a network partitioning issue. To avoid such an undesirable scenario, Aloe uses a novel approach where the μCs are encapsulated inside a container and deployed as a service inside the infrastructure nodes itself. Thus Aloe supports *μC as a Service* (μCaaS), which ensures fault-tolerance of the system. μC containers can be migrated to a target node quite easily with the help of the live-migration technique of a container when the host node fails. Aloe ensures that a set of μCs is always live in the system, maintaining the requirements for minimized switch-controller delay. On the other hand, a μC container can be customized depending on the available capacity of the nodes and resource consumptions by the controller applications. It can be noted that this μC architecture is different from existing distributed SDN controller approaches, such as DevoFlow [29] and SCL [30], which require switch-level customization. μCs can run over the existing COTS devices without any requirement for switch-level modifications.

IV. Design of Aloe Orchestration Framework

This section discusses the Aloe orchestration framework by highlighting various functional modules of Aloe and their working principles. Finally, we develop a set of APIs for language-independent and robust deployment of applications over the Aloe framework. The various functional modules of Aloe and list of acronyms used are shown in Fig. 2 and Table II respectively; the detailed description follows.

A. Aloe Functional Modules

The proposed framework consists of four node-level modules and one SNC-level module. The node-level modules run inside the infrastructure nodes and decide the topology and service parameters that need to be synchronized across various nodes. These modules collaborate with each other to take distributed decisions in a fault-tolerant way. It can be noted that in Aloe, infrastructure nodes are mutable and they can convert themselves as a μC if required. An interesting feature of Aloe is that this decision mechanism is executed in a pure distributed way, preserving the safety and liveness of the system in the presence of faults. The functionalities of various modules are as follows.

1) Topology Management Module (TMM): We design Aloe as a plug-and-play service, where an Aloe-supported IoT device can be directly deployed in an existing system for flexible auto-scaling support. The TMM initializes the Aloe framework on a newly deployed node. The tasks of the TMM are as follows – (i) identify the nodes in the neighborhood, and (ii) determine whether an Aloe service is running in that node. An Aloe service is of two types – (a) μC service, and (b) user application service. To find out the active nodes in the neighborhood, TMM uses *Link Layer Discovery Protocol* (LLDP) which is a standard practice for SDN controllers. We assume that each Aloe service deployed in the IoT cloud uses a unique predefined port address. TMM queries about the services in the local neighborhood via issuing a `telnet open port` requests. Apart from the initialization, this module is invoked on detecting a failure of a node/link or μC.

2) State Discovery Module (SDM): In case of a node or a link failure after the initialization through TMM, there is a possibility that the infrastructure nodes get disconnected from the μC. To identify such a scenario, Aloe maintains various state variables for each node as follows. (i) *Controller State* (CTLR): This state variable decides whether a node is in a `general` (does not host a μC service), ``$\mu$C`` (hosts a μC) or `undecided` (an intermediate state between the general state and the μC state) state. (ii) *Priority* (PRIO): This state variable is required only if the node is undecided and denotes the priority of the node for becoming a μC. The states associated to nodes are kept and managed by the nodes themselves. However, a node can access a copy of the states from its neighbor to decide its state. SDM is responsible for accumulating the state information collected from the neighbors. SDM uses `REST` for this purpose. Once a failure event occurs, TMM invokes the SDM. SDM keeps on executing periodically until the node finds at least one μC in its neighborhood. The periodicity of the execution of this module is dependent on the link delay. For implementation purpose, we consider the periodicity as the largest delay observed to fetch data from a neighbor. The above functioning is different than control plane state discovery module which runs inside the μC and keeps track of the network states. In contrast to that, the proposed SDM keeps track of the roles (i.e. acting as μC or not) that a node is playing in its immediate neighborhood. The detailed state transitions are given in Section A

3) μC Placement Module (μPM): Based on the neighbor states collected through SDM, every node independently determines whether it needs to launch a μC service. This is done through the μPM module. We consider the nodes as the vertices of a graph where the edges determined by the connectivity between two nodes, and place the μC services to the nodes that form a *maximal independent set* (MIS) on that graph. An MIS based μC placement ensures that there would be a μC at least in one-hop distance from each node, which can take care of the configurations and flow-initiations for the application services running on that node. As we have claimed earlier and will show in § VI that the μCs utilized in Aloe are significantly light-weight but efficient for performing network and service management activities. Therefore the total overhead due to MIS based μC placement is not significant. For identification of a suitable set of μC capable nodes, we develop a distributed randomized MIS algorithm given in Algorithm 1. The novelties of this algorithm are as follows. (1) **Randomized**: The algorithm selects different nodes at different rounds, ensuring that the load for μC service hosting is distributed across the network and does not get concentrated on some selected nodes. (2) **Bounded set**: The number of deployed μCs are always bounded based on the total number of nodes in the network. (3) **Self-stabilized**: The algorithm is self-stabilized [32] and converges in linear time ensuring fault-tolerance of the system under single or multiple simultaneous faults until complete network partition occurs. The proofs of these properties are provided in the Section A.

Algorithm 1: μPM Controller Placement Algorithm

Input: B: Any arbitrary large value greater than maximum degree of the network.

```
1  Function Trial():
       /* Breaks priority ties                                   */
2      PRIO← ⌈Rand()/B⌉;
3      return;
4  Function NeighborμC():
5      if Another μC in one-hop neighborhood then
6          return true
7      else
8          return false
9  Function UMPriority:
       /* If node has unique maximum priority                    */
10     if PRIO of this node > maximum PRIO in neighborhood then
11         return true
12     else if PRIO of this node = maximum PRIO in neighborhood then
13         return false
14     else
15         return None
16 Function Main():
17     while state change detected do
18         if CTLR=general & NeighborμC()=false then
               // No μC in neighborhood
19             CTLR← undecided;
20             Trial();
               // Initialize priority
21         else if CTLR=μC & NeighborμC()=true then
               /* Two μCs are adjacent                           */
22             CTLR← general;
23         else if CTLR=undecided & NeighborμC()=true then
               /* μC found in neighborhood                       */
24             CTLR← general;
25         else
26             if UMPriority()=None then
                   /* Executor is not maximum                    */
27                 continue;
28             else if UMPriority()=true then
                   /* Executor has unique maximum
                      priority, no need for further
                      trial.                                     */
29                 CTLR← μC;
30             else
                   /* Executor has maximum but not
                      unique priority                            */
31                 CTLR← undecided;
                   /* Next round of trial starts                 */
32                 Trial();
33     return
```

4) μC Manager Module (μMM): Once a node decides its state through μPM, the μMM module initiates the μC service on the selected nodes and establishes a controller-switch relationship between the μC and the nodes with *general*

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.3008426, IEEE Transactions on Network and Service Management

state in the one-hop neighborhood. As we mentioned earlier, a μC is initiated as a containerized service over the node designated for hosting a μC by the μPM algorithm. For a node with *general* state, this process may involve changing of controller services from one μC to another μC, which requires the reestablishment of the controller-switch relationship. For this purpose, the SDN flow tables need to be migrated from the old μC to the newly associated μC. The flow table migration mechanism is specific to the SDN controller software used, and will be discussed in §V.

5) PushToNode Module (P2NM): Along with fault-tolerance, Aloe supports rapid deployment and runtime customization of the system. To implement this feature, we develop P2NM. Unlike the rest of the modules, P2NM is centralized and/or deployed in the SNC. It provides an interface for monitoring and changing the policy level information for the μC at runtime which is useful for system administrators. Aloe supported policy level information include (i) ACLs, (ii) controller application to be executed in the μC, (iii) routing protocols running in the μC, and (iv) SDM update frequency. Apart from the specified policies, Aloe also gives freedom to its user to customize the Aloe modules itself. This feature is achieved by developing a set of APIs as discussed next.

B. Application Programmer's Interfaces (API)

The primary objective of this orchestration framework is to deploy the *controller as a service* to the in-network processing infrastructure in the form of a μC. There are some significant differences between a user application service and a μC service, which makes the deployment of the later non-trivial. Unlike many user application services, performance of the management is dependent on the topological position of the μC services. A location transparent deployment of μC might allocate all the μCs in the same node if the node has sufficient resources. Such placement can degrade the network performance of the infrastructure. However, our placement algorithm is not an optimal solution. Therefore, during the design of Aloe, we consider the extendibility of this work. Many of the implemented functionalities of this framework can be reused as API for distributed controller application development. For ease of understanding, we only provide the python sample programs here. However, all the APIs can be invoked as `bash` shell commands over the SNC using P2NM.

1) Topology Monitor: Using this API, Aloe can detect a topology change event (`TopologyMonitor()`) and take actions accordingly. This API can also be used for general purpose routing application, as given in the following code.

```
''' Find shortest path between dpidS and dpidD'''
G=TopologyMonitor()
path_dpid_list=FindShortestPath(G,"delay")
```

Listing 1: Topology Change Detector

2) Distributed State Inspector: We develop this API to observe the state of the nodes (`getNeighborStates()`), which helps in developing new placement algorithms for μPM. This API relies on a remote procedure call (`rpc`).

```
''' Find max priority amongst neighbor'''
states=getNeighborStates()
maxPrioUndecided=max([v["Prio"] for v in states.
    values() if state["CTLR"]=="undecided"])
```

Listing 2: Distributed State Inspector

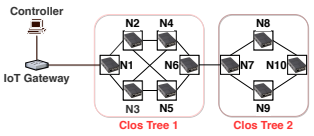


Fig. 3: Testbed:Topology

Fig. 4: Testbed:In Action

3) Find Node Services: The framework requires to identify the deployed services (`getNeighborServices()`) to enforce service level policies. We provide a python API to ease this task. The following example can be used for selective service blocking ACL.

```
'''Service blocking (ACL)'''
services=getNeighborServices()
bport=blocking_port
if(blocking_port in service["dpid"]):
 Execute("ovs-ofctl add-flow match:src=dpid,
   tcp_port=bport action:drop")
```

Listing 3: Find Node Services

Next, we discuss the details of the Aloe implementation as a general orchestration framework.

V. Aloe Implementation

We have implemented Aloe as a middleware over Linux kernel with the integration of open-source technologies, like docker containers, various SDN controllers, and `REST` based communication modules. The implementation details are described as follows.

A. Environmental Setup

We deploy an in-house testbed using the topology given in Fig. 3 for performance analysis of Aloe. The testbed follows clos tree based topology and spans across two different sites to resemble the topology given in [33]. The nodes in the testbed are Raspberry Pi version 3 Model B. The nodes are connected via multiple 100Mbps USB-to-Ethernet adapter-edges representing the physical Ethernet links among the nodes. Each links are configured to have 5Mbps of bandwidth and $100ms$ of propagation delay to match real life IoT deployment specification. Further, to analyze the scalability of Aloe, we have also deployed Aloe in a large-scale 68-node testbed using Amazon Web Services (AWS). For this purpose, we consider a sub-topology from `rocketfuel` [34] topology which consist 68 nodes. The nodes in the topology are deployed using 18 AWS *nano* instances (1 vCPU and 512 MB RAM) and 50 AWS *micro* instances (1 vCPU and 1 GB RAM). The AWS nodes are configured with Ubuntu 16.10 operating system with Debian kernel version 4.4.0. To emulate edges between the nodes, we use the *Layer 2 Tunneling Protocol* (l2tp) between the AWS instances. Every infrastructure node, both in the testbed and in the AWS, are configured with *OVS*.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.3008426, IEEE Transactions on Network and Service Management

B. Implementation Aspects

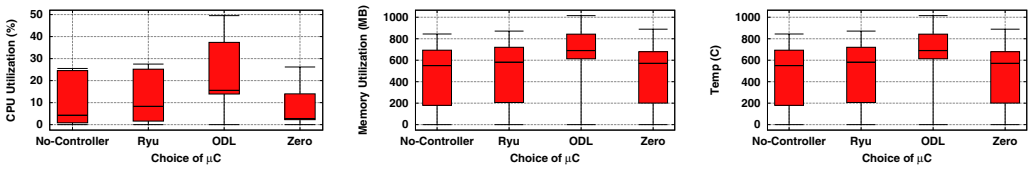
Here we discuss two important implementation aspects of Aloe – (i) flow-table consistency preservation during μC migration, and (ii) choice of controller service for μC implementation.

1) Migration of μC and consistency preservation: Due to changes in topology, node/link failure, network partition, or update in SNC policy, Aloe may require controller migration followed by a change in the node-controller association. We implement a `rpc` and `REST` based API (`changeCtlr()`) which can dynamically change a switch's allegiance from a source μC to target μC. This API can be invoked by either switch or participating μCs. `changeCtlr()` forces the switch to invoke a "`controller re-association request`" to the target μC with its previous μC address. After receiving a "`controller re-association request`", the target controller invokes migration of flow entries from the source μC. At the beginning of the migration procedure, Aloe preserves snapshots of the source μC flow table entries by sending `REST` queries to the source μC. To make the migration process lightweight, the container instance is not transferred from one node to another node; instead, the source μC container is terminated, and a new container is invoked at the target μC via `rpc`. In case of a network partitioning between the previous μC and the target μC, the target μC obtains the copy of flow table from the corresponding switch itself. In case of network partitioning between the source and the target μC, the target μC retrieves flow tables from the switches only. Since, during the migration procedure, new flow setup is deferred, the flow tables of the switches are unmodified. On the other hand, to tackle the inconsistency of neighbourhood information aroused due to the topology change, TMM updates the topology information eventually. In this way, the μMM preserves weak consistency of the system. Once the migration process is complete, the deferred flows are resumed. Due to the lower migration time, flow termination during the deferred period is negligible.

2) Choice of controller service for μC: The efficiency of Aloe is dependent on the efficiency of the choice of a controller service for the μC. Deployment of a heavy-weight controller can over-consume resources of the nodes; moreover, one μC is only responsible for managing a small set of nodes. Therefore, we target to opt for a light-weight μC for Aloe. In order to identify a suitable controller platform for μC, we compare a set of existing SDN controller services like "`Open Day light (ODL)`" [35], "`ONOS`" [19], "`ryu`" [36] and "`Zero`" [37] in our in-house testbed in terms of theirs resource utilization. Amongst these controllers, "`ONOS`" requires high memory consumption ($> 500MB$) which creates an instability in the docker environment. Further, we have observed that approximately 32% times, "`ONOS`" fails to execute over the testbed nodes due to unavailability of sufficient virtual memory. Therefore, we report the performance of the controllers other than "`ONOS`". The performance is reported based on three major system parameters – CPU utilization, memory utilization and CPU temperature variation by generating 45 flows randomly in the system using Python `SimpleHTTPServer`. In Fig. 5a, we provide the comparison of the performance of the competing controllers in terms of CPU utilization. We observe that approximately 30% "`ODL`" μCs use more than 30% of the CPU utilization. In comparison to that, around 40% "`Zero`" μCs use 15% of the CPU utilization. In Fig. 5b, we observe that almost 80% "`ODL`" μCs use more than $600MB$ of memory space. All the other controllers show lower memory utilization. Fig. 5c shows the variation in CPU core temperature while executing different types of controller services. The consolidated pair-wise comparison of the controllers are provided in Table III (upper right triangle in blue color). The notation **X** signifies that the comparison cannot be obtained. On the other hand an upper arrow ($\uparrow$) suggests that the μC listed in the top header consumes less amount of resources. We use $\leftarrow$to denote the higher efficiency of the μC application mentioned in the left header. To ascertain the comparison, we perform a statistical hypothesis testing using non-parametric, one-tailed Wilcoxon rank sum test ($\alpha = 0.01$) [38]. Our alternative hypothesis assumes that the mean resource consumptions and the core temperature is higher than the one in the normal case. The left lower triangular part of Table III (given in green color) signifies the p-values obtained

TABLE III: Wilcoxon Rank Sum Test ($\uparrow$indicates μC in top header consumes less resources, $\leftarrow$indicates μC in left header consumes less resources, **X** indicates the choice is undetermined)

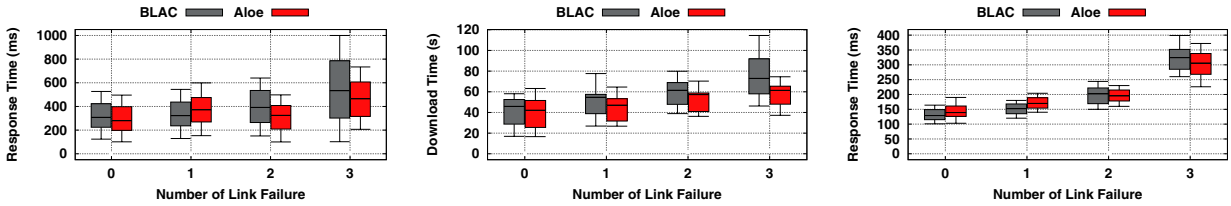
CPU				
μC	**No μC**	**Ryu**	**Zero**	**ODL**
No μC		$\leftarrow$	$\leftarrow$	$\leftarrow$
Ryu	< 0.0001		$\uparrow$	$\leftarrow$
Zero	< 0.0001	< 0.0001		$\leftarrow$
ODL	< 0.0001	< 0.0001	< 0.0001	
Memory				
No μC		$\leftarrow$	**X**	$\leftarrow$
Ryu	< 0.0001		**X**	$\leftarrow$
Zero	> 0.03	> 0.01		$\leftarrow$
ODL	< 0.0001	< 0.0001	< 0.0001	
CPU Temperature				
No μC		**X**	$\leftarrow$	$\leftarrow$
Ryu	> 0.39		$\uparrow$	$\leftarrow$
Zero	< 0.0001	< 0.0001		$\uparrow$
ODL	< 0.0001	< 0.0001	< 0.0001	



(a) CPU Utilization (b) Memory Utilization (c) CPU Temperature Variation

Fig. 5: Resource Utilization Comparison of Controller Applications

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.3008426, IEEE Transactions on Network and Service Management



(a) Cassandra (b) HTTP Server (c) Gluster

Fig. 6: Comparison of response time of services obtained from testbed: Average percentage improvement of Aloe – (a) HTTP server: 4% (0-fail), 11% (2-fails), 21% (3-fails), (b) Cassandra: 9% (0-fail), 26% (2-fails), 37% (3-fails), (c) Gluster: -8% (0-fail), 0.1% (2-fails), 6% (3-fails)

TABLE IV: Wilcoxon Rank Sum Test conclusions with p-values over response time of different applications:
X=Inconclusive, ✓=*Aloe* better, •=In band better

Services / Failure	Cassandra	HTTP	Gluster
0	**X (>0.11)**	**X (>0.20)**	**•(<0.0001)**
1	**✓(<0.0001)**	**X (>0.04)**	**•(<0.0001)**
2	**✓(<0.0001)**	**✓(<0.0001)**	**X (>0.39)**
3	**✓(<0.0001)**	**✓(<0.0001)**	**✓(<0.01)**

from the rank test. Our experimental results suggest that, "`Zero`" requires less CPU utilisation, less CPU temperature than almost all the competitors (except "`ODL`"). The memory utilisation of "`Zero`" is indistinguishable to that of "`Ryu`". "`Zero`"'s micro-kernel architecture is responsible for the resource requirement benefits. Therefore, we use "`Zero`" as our choice of μC in both testbed and AWS.

With this implementation, we evaluate the performance of Aloe, as discussed in the next section.

VI. Evaluation

We have tested the performance of Aloe with three different categories of standard applications which are common and useful for an IoT based platform – (i) HTTP service (Python `SimpleHTTPServer`): used for bulk data transfer via web clients, (ii) distributed database service (`Cassandra`): for data-driven applications, and (iii) distributed file system service (`Gluster`): used for file sharing and fault-tolerant file replication over a distributed platform. We further compare the performance of Aloe with BLAC [27], a distributed SDN control platform. To emulate realistic fault models in the system, we have injected the faults using Netflix *Chaos Monkey* fault injection tool. We have taken the measurements under all possible link fault combinations [2] in the testbed and 100 different random fault combinations in AWS[3].

A. Application Performance

Fig. 6b compares the download time of a 512MB file hosted using the HTTP service under the influence of both BLAC and Aloe over the in-house testbed. The results are obtained by varying all possible source-destination pairs in the topology. We observe that, even though Aloe results in higher download time compared to BLAC when there is no failure in the system, the performance improves rapidly in the presence of link outage. While injecting failure, we observe that approximately 30% connections are timed-out while operating under the governance of the BLAC controller. However, Aloe reduces such flow termination[4] ($< 5\%$ connection time-out for Aloe). To compare the differences of the nature of the results for each service, we performed a Wilcoxon rank sum test. The p-values and conclusion from the test is summarised in Table IV.

Fig. 6a shows the response time of `Cassandra` search queries. Here, we observe a significant difference in the characteristics of the plots due to the nature of the service. Unlike `HTTP`, `Cassandra` utilizes short flows to fetch query results. Therefore, we observe that Aloe provides a significant improvement in the query response time. However, in case of `Gluster`, Aloe performance is marginally poor compared to BLAC until there are 3 link failures (Fig. 6c). `Gluster` flows are short-distant flows, usually within one-hop. The flow-setup delay is almost negligible for a one-hop flow. Therefore the μC deployment overhead of Aloe is more when the number of failures is less.

Similar behaviors are observed in the large-scale deployment of Aloe in the AWS cloud. In Figs. 7b and 7c, `HTTP` and `Gluster` response times show similar characteristics as observed in the testbed. In the case of `Cassandra` (Fig. 7a), all the cases perform significantly better than BLAC. From these observations, we conclude that Aloe performs significantly better for the services that generate long-distant mice flows (like database synchronization). For a long-distant flow, the flow setup delay is high, which gets further affected by link failures. As a consequence, Aloe performance is better for failure-prone systems, like IoT clouds, as the flow-setup delay gets increased with the recovery time due to a failure.

B. Dissecting Aloe

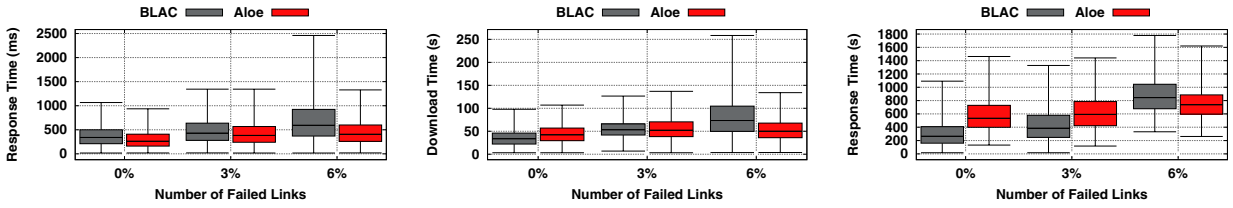
Aloe flow-setup time is dependent upon the convergence time of μPM and the path restoration time. Fig. 8a shows the distribution of the average convergence time of Aloe in the presence of failure. We have an interesting observation here that as the number of simultaneous failures increases, the convergence time drops. This can be explained as follows. Let us consider two different faults. If the two faults are at two different sides of the network, then two waves of μPM

[2]A node failure is equivalent to simultaneous failure of multiple links. Therefore, all possible link failure automatically covers node failure scenarios.

[3]All experiments are repeated 3 times

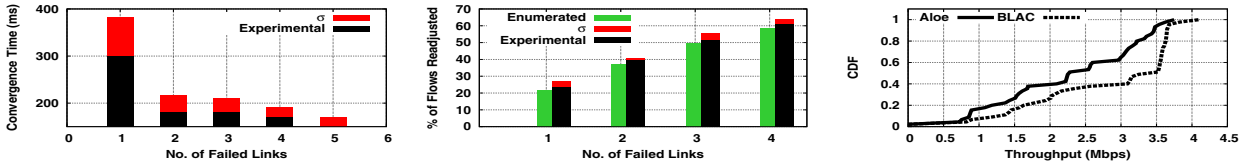
[4]Only in such cases, where the network is partitioned

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.3008426, IEEE Transactions on Network and Service Management



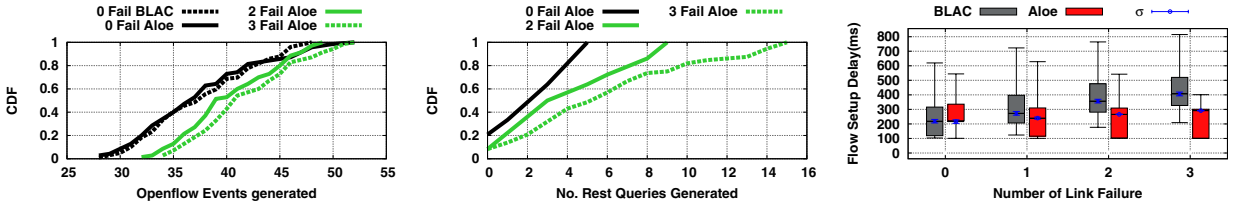
(a) Cassandra (b) HTTP Server (c) Gluster

Fig. 7: Comparison of response time of services obtained from AWS cloud: Average percentage improvement of Aloe – (a) HTTP server: -2% (0-fail), 0.1% (2%-fails), 34% (6%-fails), (b) Cassandra: 20% (0-fail), 21% (2%-fails), 34% (6%-fails), (c) Gluster: -12% (0-fail), -6% (2%-fails), 14% (6%-fails)



(a) Avg. Aloe Convergence Time (b) Avg. No. Flow Restoration (c) Throughput comparison

Fig. 8: Testbed: Effect of failure on Aloe performance (σ= standard deviation)



(a) Control Plane Overhead (b) `REST` queries generated in Aloe (c) Flow Setup Delay:using `ping`

Fig. 9: Testbed: Comparison of Aloe overhead and Flow setup delay

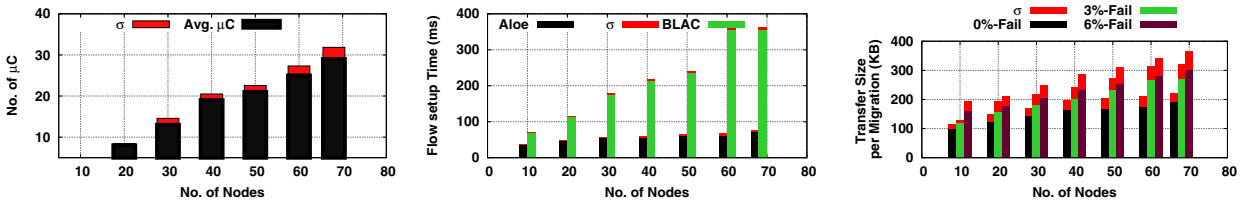
starts executing simultaneously from two different ends of the network. These two waves get diffused in the network and meet in the middle of the network at convergence. That way, multiple faults create multiple such μPM waves in the network in parallel, and as these individual waves need to deal with a smaller part of the network, they converge quickly.

The convergence phase is followed by the path restoration due to a change of the controller positions in case of a failure. To identify the performance of the path restoration, we measure the average number of flow adjustments done by the framework. The number of flow adjustment depends on the topology and the number of flows passing through the failed links. Therefore, to compare the result, we provide the enumerated number of flow adjustment required for all possible cases of link failures in Fig. 8b. We observe that the experimental observations closely match with the results obtained by enumeration. Further, these metrics have a direct impact on the flow-setup delay. To understand the effect of these factors, we compare the flow-setup delay for BLAC control plane and Aloe. To identify the flow-setup delay, we use `ping` to transfer a single `ICMP` packet. Fig. 9c shows that although Aloe marginally increases the flow setup-delay in the absence of a failure, it provides quick flow-setup when multiple faults occur in the network.

We observe that the overhead of distributed μC in Aloe is responsible for the increase in the flow-setup delay during the no-failure scenario. However, it is difficult to compare the exact overhead of the BLAC control plane and Aloe due to the differences in the nature of the overhead. We measure the overhead with respect to two different factors. Fig. 9a shows the comparison between BLAC and Aloe regarding the number of `openflow events` generated over a period of 100s. Aloe additionally generates `REST` queries to support inter-controller communication, therefore it has more number of openflow events compared to BLAC. Fig. 9b depicts the number of `REST` queries generated in Aloe. During the failure events, Aloe μC may need to migrate from one node to another. Fig. 10c shows the data transfer overhead required for migration, which is in the order of a few KB. As the number of nodes in the IoT environment are increased, the number of flow table entries are also increased. Therefore, the transfer size per migration also increases when the number of nodes are increased. The size of the flow table entries also increases with more number of failures in the network, which introduces some of redundant flow entries ("zombie flows"). However, we observe that, the effect of redundant flows has marginal effect when the number of nodes in the system are significantly high. Due to these overheads, Aloe incurs higher communication overhead than the BLAC control plane. However, due to the significant reduction in flow-setup time, Aloe ensures better flow throughput than BLAC, as shown in Fig. 8c.

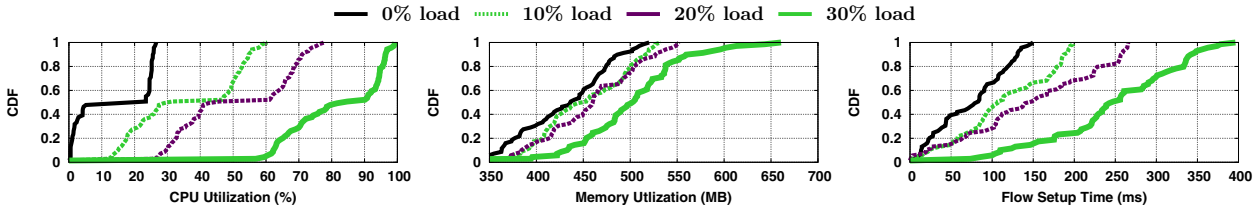
Although Aloe incurs communication overhead, Aloe en-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.3008426, IEEE Transactions on Network and Service Management



(a) Avg. No. of μC deployed (b) Avg. Flow Setup Time (c) Avg. Controller Migration Overhead

Fig. 10: Effect of Scaling Aloe μC Deployments (σ= standard deviation)



(a) CPU Utilization (b) Memory Utilization (c) Flow Setup Time

Fig. 11: Effect of Application Workload of the Host Devices on Aloe Performance

sures a significant drop in the average flow-setup delay. To limit the flow-setup delay, Aloe provides elastic auto-scaling by increasing the number of μC instances to guarantee that each node can find a μC in its neighborhood. Fig. 10a shows the average number of μC instances when the network scales, as obtained from the AWS implementation. The effect of elastic auto-scaling is shown in Fig. 10b[5] which indicates that the flow-setup delay only increases marginally in comparison to the BLAC controller, which incurs a significantly high flow-setup delay as the number of nodes in the network increases.

VII. Aloe Performance Optimization

Aloe μCs are deployed over existing network infrastructure which may have their own workloads due to the application services deployed in them; we call them as the application workloads of the devices. We perform a pilot study to check the impact of application workload of the devices on Aloe performance. We use the same AWS cloud-based deployment of Aloe as discussed earlier. Fig. 11 shows the impact of application workload on Aloe performance. To increase the application workload of the devices, we use "`stress-ng`" [39] tool. During the experiments, memory and CPU utilization of the devices have been increased from 0% to 30% of the actual capacity. When the application workload of the devices are increased, the system resources get over-utilized due to thrashing. Therefore, the system performance reduces aggressively as the μC receives less CPU time. Additionally more swap events are generated which increases the flow setup time, as we observe from Fig. 11c. These observations confirms that, Aloe μCs require special attention in terms of resource reservation for severely loaded systems. We accordingly develop a resource management framework for Aloe, which is discussed next.

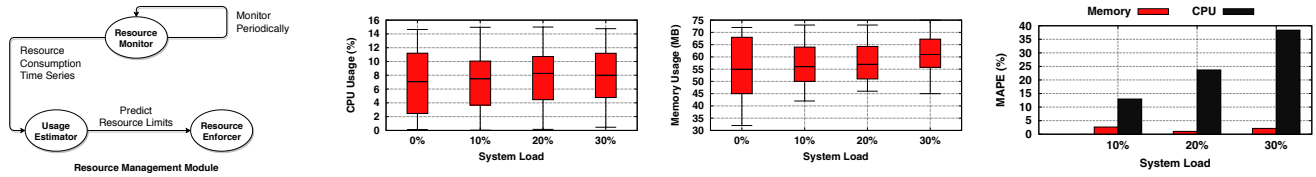
[5]Each experiments are repeated atleast 10 times

A. Effect of Resource Reservation

Reserving resources for μC applications ensures QoS in terms of flow setup time. To optimize the performance of the system, the resource reservation must match the resource demand of the μC. However, the resource demands of a μC at a particular time depends on the amount of flows managed by that μC. Therefore, we assume that the resource demand of a μC follows a temporal pattern and depends on the network state of the IoT infrastructure. Although over-provisioning resources to the μC improves the QoS, it might affect the primary workload of the devices; therefore, it can have negative impact on the overall application performance. Consequently, we implement a *Resource Management Module* (RMM) based on *Monitor-Forecast-Adapt* strategy which gets executed in each μC to balance the μC resource demands and the primary workloads of the devices. Fig. 12a shows the components of Aloe RMM which consists of three sub-modules. a) *Resource Monitor* periodically collects the usage statistics of the μC and stores it in a `JSON` data-store. b) *Usage Estimator* periodically analyzes the time-series of the resource usage pattern of the μC and predicts the probable resource demand for the next time period. c) *Resource Enforcer* is responsible for actually resource reservation for the μC based on the predicted resource demand.

1) Prediction of μC Resource Demands: For prediction of resource demands, it is important to identify the distributions of the resources which depend on the flow arrival pattern. However, in practice, it is difficult to estimate the flow arrival distribution for a large scale IoT platform with heterogeneous applications executing in it. Therefore, we choose a forecasting model based on the characteristics of the IoT applications. We focus on two basic characteristics of the IoT applications. (i) IoT applications generate bursty and short lived flows [6]. The bursty and short living nature of IoT flows reveal that, the flow arrival rates per μC during a discrete time interval is cyclic. (ii) These characteristics also suggest that, the flow arrival

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.3008426, IEEE Transactions on Network and Service Management



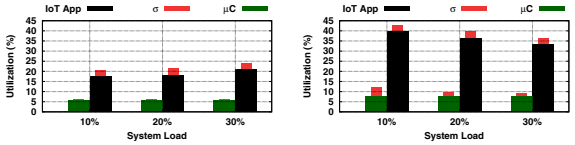
(a) Resource Management Module (b) CPU reservation (c) Memory reservation (d) MAPE Score

Fig. 12: Resource Reservation for Aloe μCs

rates follow a non-stationary property. Therefore, we use *Autoregressive Integrated Moving Average* (ARIMA) [40] model for forecasting of individual resource requirements. ARIMA relies on the mean reversion principle of non-stationary data to forecast the future strategy based on the time series by employing autoregression.

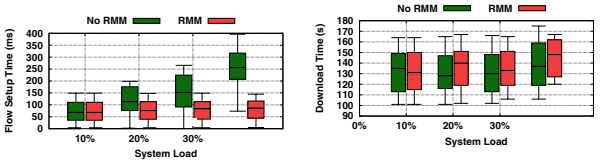
2) Performance Improvement with Aloe RMM: We have integrated the RMM module with Aloe and tested it over the AWS platform as discussed earlier. Like many statistical modeling methods, identification of parameters for ARIMA is a non-trivial challenge. Therefore, we use auto-ARIMA [40] based on our experimental observations to individually forecast the CPU and memory demands according to the resource utilization time series. Fig. 12b shows the amount of CPU reserved for the μC in various load scenarios remains almost constant. Fig. 12c shows the memory reservation of the μC application due to resource reservation module. From the results we can observe that, the memory reservation amount increases in case of 30% load. The reason behind this observation lies in the OVS to μC mapping technique used in Aloe μMM. An OVS chooses a μC based on how quickly the μC responds to its join request. As the system load increases, a lightly loaded μC is more likely to provide a quick response time. Therefore, the lightly loaded μCs are likely to get connected with more switches with high number of flows passing through those switches, which in turn increases the memory overhead of those μC. Next we check how accurate the RMM prediction model can perform based on the mean average percentage error (MAPE). Fig. 12d revels that, the proposed RMM provides significantly low MAPE for prediction of the memory. On the other hand, higher MAPE was observed for CPU utilization prediction. The reason behind this observation is fluctuation of CPU is very frequent which the underlying ARIMA can not predict always. Therefore, the performance of the IoT applications are also influenced. Figs. 13a and 13b compare the resource utilization between the μC and IoT application in terms of CPU and memory. From Fig. 13a, we observe that the accuracy of RMM does not significantly affect the performance of memory utilization by the IoT applications. However, the reduced accuracy of used ARIMA sometimes over provisions more CPU time to the μCs. As a result the IoT application receives less CPU time than its demand in such cases.

Interestingly, the IoT application (like GlusterFS) shares more host resources than the μC; therefore, a slight resource biasing towards the μCs improve their performance significantly, while having marginal impact on the performance of the IoT application. We present this observation in Fig. 14a,



(a) Memory Utilization (b) CPU Utilization

Fig. 13: Effects of Resource Reservation (σ= standard deviation)



(a) Flow setup time (b) GlusterFS file download time

Fig. 14: Effects of Resource Reservation for Aloe μCs

where we compare the performance of the RMM in terms of flow setup time with that of no-RMM. The results justifies that the RMM ensures low variations in flow setup time as opposed to the no-RMM case. In fact use of RMM can significantly improve the performance of IoT short flows by reducing average flow set-up time by $13\% - 120\%$ in various load scenarios. However, due to the resource reservation of μC, the application may suffer due to insufficient resources. To understand this effect, we compare the performance of the GlusterFS application before and after implementing the RMM in Fig. 14b. We find that, the increase in mean download time due to effects of RMM while downloading a $25MB$ file using GlusterFS varies between $2\% - 7\%$ for different load conditions which is considerably small.

VIII. Conclusion

In this paper, we present Aloe, an orchestration framework, for IoT in-network processing infrastructures. Aloe uses docker container to support lightweight migration capable in-band controllers. This design choice helps Aloe to provide elastic auto-scaling while keeping flow setup time under control. Aloe provides controller as a service to exploit in-network processing infrastructure and supports fault and partition tolerance. The performance of Aloe has been tested thoroughly and compared with a very recent orchestration framework (BLAC). The results indicate a significant improvement in response times for distributed IoT services. The low-level system properties of the Aloe framework can be validated through formal modeling of the Aloe functional algorithms, which can also provide the formal guarantee of the system performance. We keep this as a future direction of this work.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/TNSM.2020.3008426, IEEE Transactions on Network and Service Management

ACKNOWLEDGMENT

This work was supported by Science and Engineering Research Board (SERB) Early Career Research Award, File number: ECR/2017/000121 Dated 18/07/2017, funded by Department of Science and Technology, Government of India.

APPENDIX

The core characteristics of Aloe depend upon the proposed MIS algorithm (Algorithm 1) in μPM. Here we analyze the theoretical properties of the MIS algorithm used in Aloe. For this purpose, we denote the topology of the in-network processing infrastructure as a undirected and unweighted graph $G = (V, E)$, where V represents the set of in-network processing nodes and E represents the communication links between them. We define a node v as the neighbor of another node u if there exists a communication link between u and v. The state of a node u is defined as the a variable `CTLR` and is denoted as $\texttt{CTLR}_u$. As per Algorithm 1, $\texttt{CTLR}_u$ can take one of the following values – `general`,`undecided` and μC. A node goes through a state transition based on the state of its neighborhood and the value of the `PRIO` variable of the closed neighborhood. For the ease of reference, we present the state transitions of a single node using Fig. 15. Based on the the individual state of the nodes, we define the term *"global state"* to represent the overall state of the platform. The global state of the platform is defined as $\mathcal{S} = (\texttt{CTLR}_u : \forall u \in V)$ is an ordered tuple of length $|V|$. We define a *"legitimate state"* as any particular global state among the all possible global states, where no further statement can be applied at any node.

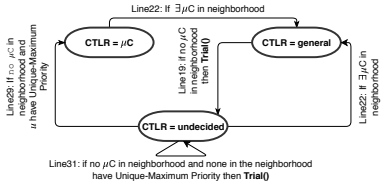


Fig. 15: State transition diagram of a node u

As mentioned earlier, we claim that the proposed algorithm is self-stabilizing. A proof of self-stabilization requires the proof of **Closure property** and **Convergence property**. The closure property of a system states that, the system will remain in a legitimate state if the system was in legitimate state and there is no perturbation in the system. On the other hand, if the convergence property is ensured by the system, the system will reach to a legitimate state in the absence of further perturbation.

A. Aloe μPM Ensures Closure Property

Theorem 1. *If any node in the IoT platform is in* `Undecided` *state then the platform executing Algorithm 1 is not in a legitimate state.*

Proof: Let us assume that, $CTLR_u =$`undecided` and the platform is in a legitimate state. In that case, only the following scenarios may occur.

Case-1 If $\exists v : CTLR_v = \mu$C and v is in the neighborhood of u. In that case, NeighborμC() returns `true` and Line 24 will be applicable.

Case-2 $\nexists v : CTLR_v = \mu$C for all v in the neighborhood of u. Let Q_u denotes the set of all v such that, each v is a neighbor of u and in `Undecided` state.

Case-2.i If u has unique maximum priority (i.e. UMPriority()=`true`) or $Q_u = \emptyset$, u executes Line 29.

Case-2.ii If u has maximum priority but not unique (i.e. UMPriority()=`false`) then, u executes Line 31.

Case-2.iii If u does not have maximum priority (i.e. UMPriority()=`None`) then, $\exists v$ which has/have maximum priority and it/they will execute either Line 29 or Line 31.

All these cases mentioned above contradicts the assumption that, the platform is in a legitimate state. Therefore, we can claim that, Algorithm 1 can not be in a legitimate state when at least one of the nodes is in the `Undecided` state. ■

Theorem 1 also proves that, if the platform is in a legitimate state, all the nodes are in either in the μC state or in the `general` state. As the system can not execute any statement in a legitimate state, then the following statements holds.

Corollary 1.1. ***(Closure property)*** *If no failure occurs, an IoT platform in "legitimate state" will remain in a legitimate state forever.*

The significance of legitimate state can be described by the theorem as follows.

Theorem 2. *If the platform is in a legitimate state, the set of μCs form a maximal independent set (MIS) of G.*

Proof: To prove this statement, we use proof by contradiction. We assume that the platform is in a legitimate state and the μCs do not form a MIS. There can be only two cases possible as per the definition of MIS.

Case-1 The set of μC does not hold the independence relationship. Therefore, $\exists u, v$ such that they are neighbor and both of them are in μC state simultaneously. In this case, Line 24 is applicable.

Case-2 The set of μC does not hold the maximality relationship. Therefore, $\exists u$ having `general` state can become μC without affecting other μCs. In this case, atleast one of the nodes can execute any one of statements given in Lines 19, 29 and 31.

Both of the cases mentioned above violates the the legitimate state assumption of the platform. ■

Theorems 1 and 2 proves that, Aloe is stable and consistent when there is no failure in the platform and after reaching legitimate state all μCs form a MIS of the original IoT platform. Thus, the flow setup delay is minimum as each switch has one μC in its vicinity.

B. Aloe Convergence after Each Failure

As a legitimate state can not accommodate any node in `undecided` state, the platform can only reach in a legitimate state once all the nodes in `undecided` state have changed their states by taking suitable transitions. So, in the following

theorem, we prove that traces of the state changes possible in individual nodes.

Theorem 3. *Only the following sequence or subsequence of state change is possible for each node during the execution of the algorithm, if no further failure occurs in between.*
(`Undecided`$\rightarrow$`general`$\rightarrow$`Undecided`$\rightarrow$`general`*)*
(`Undecided`$\rightarrow$`general`$\rightarrow$`Undecided`$\rightarrow \mu$*C),*
(μ*C*$\rightarrow$`general`$\rightarrow$`Undecided`$\rightarrow$`general`*),*
(μ*C*$\rightarrow$`general`$\rightarrow$`Undecided`$\rightarrow \mu$*C)*

Proof. As per Fig. 15, if a node u initially had μC or `general` state, then it requires at most 2 or 1 state change to reach `Undecided` state, respectively. On the other hand, if a node u executes Line 29, it will not execute any other rule unless a failure occures; and no other nodes in the neighborhood of u can become μC after that. Therefore, if u executes Line 29, its neighbors can only execute Line 22. These properties ensure the statement of Theorem 3. □

However, Theorem 3 does not ensure a bound on the number of statement execution required for a node to reach `general` or μC from `undecided` state. We prove the expected bound based on the distribution of the event of finding unique maximum value of `PRIO` variable in the close neighborhood.

Theorem 4. *If N is number of nodes in a closed neighborhood of any u, then $P(N,B)$ denote the probability of finding an unique maximum in the closed neighborhood of u. Then*

$$P(N,B) = \frac{(N \times \sum_{i=1}^{B} i^{(N-1)})}{(B+1)^N}$$

Proof. Let i be the highest priority in a configuration $\mathcal{S}$ after a round, where each round corresponds to the event of generating the priority by at most each nodes in the closed neighborhood of u. The unique maximum property suggests that, only one node has `PRIO`$= i$ and the rest of nodes can have $0 <$ `PRIO` $\leq i-1$. The value of i can vary from 1 to B. For a fixed i, the rest of `PRIO` values of the nodes can be arranged in $N \times i^{(N-1)}$ different ways. Hence the total probability calculated for a UMPriority generation event can be expressed as follows.

$$P(N,B) = \frac{(N \times \sum_{i=1}^{B} i^{(N-1)})}{(B+1)^N}$$

□

If all nodes in the closed neighborhood of u are in `Undecided` state, all of the N nodes are executing Line 29 or Line 31 as there is no μC adjacent to them. To find the expected number of rounds for one of the intermediate node to become μC, we have to find the expected number of rounds in which there will be only one node with maximum priority in the neighborhood.

Theorem 5. *If X denote the random variable indicating the number of rounds required to find a unique maximum priority in the closed neighborhood of u then $E[X] \leq e$.*

Proof. The random variable and the distribution function can be represented as follows.

$$Pr[X = r] = P(N,B)(1 - P(N,B))^{(r-1)}$$

The expected number of rounds can be calculated as Eq. (1).

$$E[X] = \frac{1}{P(N,B)} = \frac{(B+1)^N}{N \sum_{i=1}^{B} i^{N-1}} \tag{1}$$

As the value of N is upper bounded by $(B+1)$, Eq. (1) can be expressed as follows.

$$E[X] \leq \frac{(B+1)^{B+1}}{(B+1)\int_0^B i^B \, di} = \frac{(B+1)^{(B+1)}}{B^{(B+1)}} = (1+B^{-1})^{(B+1)}$$

The upper bound of $E[X]$ is attained at $B \to \infty$ and the value is calculated in Eq. (2).

$$\lim_{B \to \infty} \left(1 + B^{-1}\right)^{(B+1)} = e \tag{2}$$

Therefore, $E[X] \leq e$. This result signifies that a node and it's neighbors require expected *"e"* statement execution by each node for generation of unique maximum priority generation event. □

Theorems 3 to 5 proves that, each node is expected to execute $2+e$ statements to reach a non `undecided` state from any arbitrary state. We can also conclude that, the cumulative number statement execution of the entire platform to arrive at a a legitimate state from any arbitrary state is expected to be $\mathcal{O}(|V|)$. Hence, the proposed algorithm is linear in terms of execution complexity.

REFERENCES

[1] D. Kreutz, F. M. Ramos, P. E. Verissimo, C. E. Rothenberg, S. Azodolmolky, and S. Uhlig, "Software-defined networking: A comprehensive survey," *Proceedings of the IEEE*, pp. 14–76, 2015.

[2] A. C. Baktir, A. Ozgovde, and C. Ersoy, "How can edge computing benefit from software-defined networking: a survey, use cases, and future directions," *IEEE Communication Surveys & Tutorials*, pp. 2359–2391, 2017.

[3] L. M. Vaquero and L. Rodero-Merino, "Finding your way in the fog: Towards a comprehensive definition of fog computing," *ACM SIGCOMM Computer Communication Review*, pp. 27–32, 2014.

[4] M. Chiang, S. Ha, I. Chih-Lin, F. Risso, and T. Zhang, "Clarifying fog computing and networking: 10 questions and answers," *IEEE Communication Magazine*, pp. 18–20, 2017.

[5] M. Selimi, L. Cerdà-Alabern, M. Sánchez-Artigas, F. Freitag, and L. Veiga, "Practical service placement approach for microservices architecture," in *Proceedings of the 17th IEEE/ACM International Symposium in Cluster, Cloud, and Grid Computing (CCGRID)*, 2017, pp. 401–410.

[6] M. Z. Shafiq, L. Ji, A. X. Liu, J. Pang, and J. Wang, "Large-Scale Measurement and Characterization of Cellular Machine-to-Machine Traffic," *IEEE/ACM Transaction on Networking*, pp. 1960–1973, 2013.

[7] O. Kaiwartya, A. H. Abdullah, Y. Cao, J. Lloret, S. Kumar, R. R. Shah, M. Prasad, and S. Prakash, "Virtualization in wireless sensor networks: fault tolerant embedding for internet of things," *IEEE Internet of Things Journal*, pp. 571–580, 2018.

[8] L. Galluccio, S. Milardo, G. Morabito, and S. Palazzo, "SDN-WISE: Design, prototyping and experimentation of a stateful SDN solution for wireless sensor networks," in *Proceedings of the 34th IEEE International Conference on Computer Communications (INFOCOM)*, 2015, pp. 513–521.

[9] F. Tang, Z. M. Fadlullah, B. Mao, and N. Kato, "An intelligent traffic load prediction based adaptive channel assignment algorithm in SDN-IoT: A deep learning approach," *IEEE Internet of Things Journal*, 2018.

[10] D. Wu, D. I. Arkhipov, E. Asmare, Z. Qin, and J. A. McCann, "UbiFlow: Mobility management in urban-scale software defined IoT," in *proceedings of the 34th IEEE International Conference on Computer Communications (INFOCOM)*, 2015, pp. 208–216.

[11] M. T. I. ul Huque, W. Si, G. Jourjon, and V. Gramoli, "Large-scale dynamic controller placement," *IEEE Transactions on Network and Service Management*, pp. 63–76, 2017.

[12] L. Ma, S. Yi, and Q. Li, "Efficient service handoff across edge servers via docker container migration," in *Proceedings of the 2nd ACM/IEEE Symposium on Edge Computing (SEC)*, 2017, p. 11.

[13] S. Chattopadhyay, S. Chatterjee, S. Nandi, and S. Chakraborty, "Aloe: An elastic auto-scaled and self-stabilized orchestration framework for iot applications," in *Proceedings of the 39th International Conference on Computer Communications (INFOCOM)*, April 2019.

[14] K. Phemius, M. Bouet, and J. Leguay, "Disco: Distributed multi-domain sdn controllers," in *Procedeeings of 12th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, May 2014, pp. 1–4.

[15] M. A. Santos, B. A. Nunes, K. Obraczka, T. Turletti, B. T. De Oliveira, and C. B. Margi, "Decentralizing SDN's control plane," in *Proceedings of 39th IEEE Conference on Local Computer Networks (LCN)*. IEEE, 2014, pp. 402–405.

[16] F. Bannour, S. Souihi, and A. Mellouk, "Distributed sdn control: Survey, taxonomy, and challenges," *IEEE Communications Surveys & Tutorials*, vol. 20, no. 1, pp. 333–354, 2018.

[17] A. Tootoonchian and Y. Ganjali, "Hyperflow: A distributed control plane for openflow," in *Proceedings of the 2010 Internet Network Management Workshop/Workshop on Research on Enterprise Networking*. Berkeley, CA, USA: USENIX Association, 2010, pp. 3–3.

[18] T. Koponen, M. Casado, N. Gude, J. Stribling, L. Poutievski, M. Zhu, R. Ramanathan, Y. Iwata, H. Inoue, T. Hama, and S. Shenker, "Onix: A distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX Symposium on Operating Systems Design and Implementation (OSDI)*. USENIX Association, 2010, pp. 1–6.

[19] P. Berde, M. Gerola, J. Hart, Y. Higuchi, M. Kobayashi, T. Koide, B. Lantz, B. O'Connor, P. Radoslavov, W. Snow *et al.*, "Onos: towards an open, distributed sdn os," in *Proceedings of the 3rd Workshop on Hot topics in Software Defined Networking (HotSDN)*. ACM, 2014, pp. 1–6.

[20] A. Dixit, F. Hao, S. Mukherjee, T. Lakshman, and R. R. Kompella, "Elasticon: an elastic distributed sdn controller," in *Proceedings of the 10th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS)*. ACM, 2014, pp. 17–28.

[21] S. Hassas Yeganeh and Y. Ganjali, "Kandoo: A framework for efficient and scalable offloading of control applications," in *Proceedings of the 1st Workshop on Hot topics in Software Defined Networking (HotSDN)*, 2012, pp. 19–24.

[22] B. Grkemli, S. Tatlcolu, A. M. Tekalp, S. Civanlar, and E. Lokman, "Dynamic control plane for sdn at scale," *IEEE Journal on Selected Areas in Communications*, vol. 36, no. 12, pp. 2688–2701, Dec 2018.

[23] U. Siddique, K. A. Hoque, and T. T. Johnson, "Formal specification and dependability analysis of optical communication networks," in *Design, Automation Test in Europe Conference Exhibition (DATE), 2017*, 2017, pp. 1564–1569.

[24] L. Sidki, Y. Ben-Shimol, and A. Sadovski, "Fault tolerant mechanisms for sdn controllers," in *2016 IEEE Conference on Network Function Virtualization and Software Defined Networks (NFV-SDN)*, Nov 2016, pp. 173–178.

[25] A. Mantas and F. M. V. Ramos, "Rama: Controller fault tolerance in software-defined networking made practical," *CoRR*, vol. abs/1902.01669, 2019. [Online]. Available: http://arxiv.org/abs/1902.01669

[26] T. Hu, Z. Guo, J. Zhang, and J. Lan, "Adaptive slave controller assignment for fault-tolerant control plane in software-defined networking," in *2018 IEEE International Conference on Communications (ICC)*, 2018, pp. 1–6.

[27] V. Huang, Q. Fu, G. Chen, E. Wen, and J. Hart, "BLAC: A Bindingless Architecture for Distributed SDN Controllers," in *Proceedings of 42nd IEEE Conference on Local Computer Networks (LCN)*, 2017, pp. 146–154.

[28] M. Yu, J. Rexford, M. J. Freedman, and J. Wang, "Scalable flow-based networking with difane," *ACM SIGCOMM Computer Communication Review*, pp. 351–362, 2010.

[29] A. R. Curtis, J. C. Mogul, J. Tourrilhes, P. Yalagandula, P. Sharma, and S. Banerjee, "DevoFlow: Scaling flow management for high-performance networks," in *ACM SIGCOMM Computer Communication Review*. ACM, 2011, pp. 254–265.

[30] A. Panda, W. Zheng, X. Hu, A. Krishnamurthy, and S. Shenker, "SCL: simplifying distributed SDN control planes," in *Proceedings of the 14th UUSENIX Symposium on Networked Systems Design and Implementation (NSDI)*. USENIX Association, 2017, pp. 329–345.

[31] A. Panda, C. Scott, A. Ghodsi, T. Koponen, and S. Shenker, "CAP for networks," in *Proceedings of the 2nd Workshop on Hot topics in Software Defined Networking (HotSDN)*. ACM, 2013, pp. 91–96.

[32] S. Chattopadhyay, N. Sett, S. Nandi, and S. Chakraborty, "Flipper: Fault-tolerant distributed network management and control," in *2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, 2017, pp. 421–427.

[33] "Cisco ACI Multi-POD and Multi-Site Demystified," May 2019, [Online; accessed 10. May. 2019]. [Online]. Available: https://netdesignarena.com/index.php/2018/07/01/cisco-aci-multi-pod-and-multi-site-demystified/

[34] "UW CSE | Systems Research | Rocketfuel," May 2005, [Online; accessed 28. Jul. 2018]. [Online]. Available: https://research.cs.washington.edu/networking/rocketfuel

[35] (2018) OpenDaylight. [Online]. Available: http://www.opendaylight.org/

[36] "Ryu 4.30 documentation," May 2019, [Online; accessed 10. May. 2019]. [Online]. Available: https://ryu.readthedocs.io/en/latest/

[37] T. Kohler, F. Drr, and K. Rothermel, "Zerosdn: A highly flexible and modular architecture for full-range distribution of event-based network control," *IEEE Transactions on Network and Service Management*, vol. 15, no. 4, pp. 1207–1221, Dec 2018.

[38] F. Wilcoxon, "Individual comparisons by ranking methods," *Biometrics Bulletin*, pp. 80–83, 1945.

[39] "stress-ng - a tool to load and stress a computer system," Jun 2018, [Online; accessed 28. Jul. 2018]. [Online]. Available: http://manpages.ubuntu.com/manpages/xenial/man1/stress-ng.1.html

[40] J. G. De Gooijer and R. J. Hyndman, "25 years of time series forecasting," *International Journal of Forecasting*, vol. 3, no. 22, pp. 443–473, 2006.

Subhrendu Chattopadhyay received the B.Tech. degree from West Bengal University of Technology, Kolkata, India, and the M.Tech. degree from Indian Institute of Technology Guwahati, Assam, India. He is currently pursuing the Ph.D. degree at Indian Institute of Technology Guwahati, Assam, India. He has received a Research Fellowship from TATA Consultancy Services, India. His research interests include the area of computer networking, distributed systems, and social network analysis

Soumyajit Chatterjee joined IIT Kharagpur in 2017, as a research scholar (Doctorate Program). He received his M.Tech in Computer Science from IIT (ISM), Dhanbad in the year 2016 and B.E. from University Institute of Technology,The University of Burdwan in 2012. He also has an industry experience of one year seven months. Currently, his domain of research is mobile systems and ubiquitous computing.

Sukumar Nandi received the Ph.D. degree from Indian Institute of Technology Kharagpur, India. Currently, he is a Professor with Indian Institute of Technology Guwahati, India. His research interests include traffic engineering, wireless networks, network security, and distributed computing. He is a Senior Member of IEEE and ACM, a Fellow of the Institution of Engineers (India), and a Fellow of the Institution of Electronics and Telecommunication Engineers (India).

Sandip Chakraborty received the Ph.D. degree from Indian Institute of Technology Guwahati, Assam, India, in 2014. Currently, he is an Assistant Professor with Indian Institute of Technology Kharagpur, Kharagpur, India. His research interests include wireless networks, mobile computing, and distributed computing. He is a Member of IEEE and the Association for Computing Machinery

The Journal of Supercomputing
https://doi.org/10.1007/s11227-021-04135-2

Containerized deployment of micro-services in fog devices: a reinforcement learning-based approach

Shubha Brata Nath[1] · Subhrendu Chattopadhyay[2] · Raja Karmakar[3] · Sourav Kanti Addya[4] · Sandip Chakraborty[1] · Soumya K Ghosh[1]

Accepted: 8 October 2021

Abstract

The real power of fog computing comes when deployed under a smart environment, where the raw data sensed by the Internet of Things (IoT) devices should not cross the data boundary to preserve the privacy of the environment, yet a fast computation and the processing of the data is required. Devices like home network gateway, WiFi access points or core network switches can work as a fog device in such scenarios as its computing resources can be leveraged by the applications for data processing. However, these devices have their primary workload (like packet forwarding in a router/switch) that is time-varying and often generates spikes in the resource demand when bandwidth-hungry end-user applications, are started. In this paper, we propose *pick–test–choose*, a dynamic micro-service deployment and execution model that considers such time-varying primary workloads and workload spikes in the fog nodes. The proposed mechanism uses a reinforcement learning mechanism, *Bayesian optimization*, to decide the target fog node for an application micro-service based on its prior observation of the system's states. We implement PTC in a testbed setup and evaluate its performance. We observe that PTC performs better than four other baseline models for micro-service offloading in a fog computing framework. In the experiment with an optical character recognition service, the proposed PTC gives average response time in the range of 9.71 sec–50 sec, which is better than Foglets (24.21 sec–80.35 sec), first-fit (16.74 sec–88 sec), best-fit (11.48 sec–57.39 sec) and mobility-based method (12 sec–53 sec). A further scalability study with an emulated setup over Amazon EC2 further confirms the superiority of PTC over other baselines.

Keywords Fog computing · Container deployment · Bayesian optimization · Internet of things · Reinforcement learning

✉ Sandip Chakraborty
sandipc@cse.iitkgp.ac.in

Extended author information available on the last page of the article

Published online: 27 October 2021

1 Introduction

The concept of fog computing [9, 18] is gaining popularity in recent times due to the availability of computationally powerful network devices, like routers, WiFi access points, core switches, etc. Various recent works have focused on developing innovative applications around it, ranging from smart cities applications [12], big data processing [25], pervasive health care [1, 19], etc. In a typical fog computing architecture, the intermediate network devices' excess computing resources are used for end-user application processing purposes. Thus, it provides low network latency along with added privacy to the data. However, the response time becomes higher when the end devices like the Internet of Things (IoT) devices do not have sufficient computing resources, and hence, the computation needs to be offloaded to a secondary device. Although a large number of research works [2, 7, 10, 17, 24, 27, 31, 33, 34] have been devoted to design application offloading mechanisms for fog computing, they primarily consider the workload placement as a one-time process and fail to capture the dynamicity of the system.

In the fog computing paradigm, the application workloads are divided into micro-services that can be executed in parallel [4]. These micro-services [15, 29, 30] are placed on the fog nodes (the routers, access points and core network switches) through some sandboxing mechanism, like the virtual machines (VMs) or the containers. Such a placement mechanism needs to consider two different aspects—(1) there should be excess computing resources available at the fog nodes, and (2) the execution of the micro-service should not interfere with the primary workload of the fog nodes. Although the existing approaches, as mentioned above, consider these two factors, they fail to properly characterize the nature of the fog nodes' primary workload. Such primary workloads are typically dynamic. Consider the following example. In a smart home environment, let the IoT devices and the corresponding applications use the home gateway as a fog node. Thus, the IoT data processing tasks are offloaded on the network gateway. Now, multiple smart home appliances are connected to that gateway, and the gateway needs to route the data packets to respective subnets, which is its primary workload. Now, this gateway's primary workload observes a spike whenever high-bandwidth applications, such as a smart television, get started. During that time, the gateway might not have sufficient computation resources to execute the fog micro-services. If we still force the micro-services to get completed on that fog node, the response time will possibly get increased, resulting in a negative performance impact.

The primary advantage of fog computing comes from the fact that it provides a quicker response than the cloud as the computation happens near the device. However, unless we properly place and schedule the micro-services over the fog nodes by considering their primary workloads, the advantages might get nullified and it might result in a very high response time. The primary requirements for developing such an execution architecture are as follows. (1) The placement algorithm should not be a one-time algorithm; it should continuously monitor the primary workload and excess resources in the fog nodes and dynamically update the target node where the micro-service can execute. (2) Depending on the dynamic

execution decision, micro-services should be able to seamlessly migrate from one fog node to another to complete its execution. (3) The migration should be fast, having negligible overhead. (4) The decision of placement and migration should be very fast so that it does not contribute significantly to the overall response time. The existing mechanisms fail to capture all of these four aspects together.

We propose *pick–test–choose* (PTC) framework which can be considered as a containerized micro-service placement mechanism for the fog nodes considering their primary workloads and their impact on the micro-service execution. PTC continuously monitors the fog nodes' excess resources and the required computation environment for the micro-service workloads to complete their executions. Accordingly, it models the problem as a constrained optimization such that the response time always remains within a predefined boundary. As solving such a constrained optimization is computationally expensive, we use a reinforcement learning technology to dynamically map the micro-service execution to the fog nodes' computation platforms. To satisfy the requirement of having a quick decision, we use *Bayesian optimization* [26] (BO) that can dynamically and quickly decide the target solution. It also considers the measurement noises that may come while calculating the resource availability (of the fog nodes) and the resource requirements (of the micro-services). We have implemented PTC over a prototype testbed for thorough performance evaluation and comparison with baselines; along with that, the performance has also been evaluated in Amazon Elastic Compute Cloud (Amazon EC2) for scalability analysis. The experimental results show that PTC can minimize the response time while effectively using fog nodes' excess resources. We further analyze PTC's performance in the context of a natural language processing (NLP) application as a use case, which has been evaluated over the testbed. We observe that the proposed framework can significantly reduce the response time of the NLP application's micro-service executions.

An initial version of this work has been published in [21]. We have made a significant extension of the initial framework in this current version. (1) We have extended the theoretical framework to formulate the dynamic optimization problem based on resource availability at the fog nodes and the fog micro-services' resource demands. We have shown how a BO-based solution model can reasonably fit in the proposed optimization framework. (2) We have extended the evaluation with a detailed analysis of PTC's performance under various scenarios. We have further analyzed a use case over the PTC framework. In summary, the salient contributions of this paper are as follows.

1. We have formulated an optimization problem considering the fog micro-services' dynamic demands along with the dynamic resource availability at the fog nodes. We have shown formally that the proposed optimization is $\mathcal{NP}$-hard.
2. In order to solve the proposed optimization, we utilized a Bayesian optimization-based framework while considering the dynamicity and measurement noises that may come during the runtime of such a system.
3. We have evaluated the proposed framework thoroughly with an optical character recognition application. The proposed PTC framework is also evaluated with a

natural language processing application as a use case. These evaluations have shown the superiority of PTC compared to other baselines.
4. We have also tested the proposed PTC framework for scalability in Amazon EC2. We have observed from the experiments in Amazon EC2 that the proposed PTC framework is scalable while ensuring minimization of response time.

The rest of the paper is organized as follows. An extensive study of the existing literature is provided in Sect. 2. The proposed system architecture and design goals are discussed in Sect. 3. The micro-service placement is modeled as an optimization, which is discussed in Sect. 4. Based on the hardness of the proposed optimization, we provide a reinforcement learning-based solution of the proposed optimization, as discussed in Sect. 5. The performance evaluation of the proposed framework is provided in Sect. 6. Finally, Sect. 7 concludes the paper.

2 Related work

Various works have focused on the micro-service deployment problem in fog computing. The authors in [28] present a metaheuristic approach to minimize the application response time in edge computing for micro-service-based applications. In [24], the authors have developed a programming infrastructure for geo-distributed applications, which launches the application modules and performs the migration of these modules between fog devices as containers. These approaches primarily choose the nearest node for application deployment and therefore do not consider that the nearest node might be loaded with clients' requests. In [7], Elgamal *et al.* have proposed a dynamic programming-based algorithm to partition operations in IoT applications. The operations are placed across edge and cloud resources to minimize the completion time of the end-to-end operations. VM-based micro-service placement and migration for mobile users have been discussed in [10] where the number of applications to be placed in fog devices has been maximized while minimizing the latency cost. In [11], the authors have studied the base station association, task distribution and VM placement for cost efficient fog-based medical cyber-physical systems. The authors have minimized the communication cost along with the VM deployment cost. Li et al.[16] have proposed a resource scheduling approach in edge computing-based smart manufacturing, ensuring latency constraints. However, all of these works have considered static workload at the fog devices; therefore, the placement algorithms are modeled mostly as a one-time optimization problem solved through some efficient heuristics.

A few works in the literature have focused on application placement at the fog devices based on dynamically monitoring of resources. Yigitoglu et al.[34] have proposed a containerized application placement strategy based on the current measurement of the fog workload. They have considered a hierarchical organization of fog devices based on their resource availability and placed the application at a suitable hierarchy. Taneja et al.[31] have presented a module mapping algorithm for efficient resource utilization by placing the application modules in fog–cloud infrastructure. However, these works still use one-time scheduling of the fog workloads depending

on the immediate availability of the computing resources at the fog nodes. Although such approaches can show good performance for short term workloads, the performance severely suffers when the execution time is a bit longer. The authors in [27] have proposed an orchestration system where the micro-services are placed in fog devices if the resources are available; otherwise, the micro-service would be placed in the cloud node. The authors have also analyzed if there is a need to offload the micro-service in the cloud or queue it to one of the busy fog devices. In [35], a container-based task scheduling and reallocation mechanism are designed to optimize the number of concurrent tasks in fog computing-based smart manufacturing. These approaches are mostly based on periodic measurements and re-execution of the full optimization, which is a significant overhead for the system. In [23], the authors have developed an orchestration tool based on Kubernetes. They have extended it with self-adaptation and network-aware placement capabilities. The authors have proposed a model-based reinforcement learning solution to solve the elasticity problem. The work of [13] has presented a distributed placement policy that optimizes energy consumption at fog nodes and communication costs. They have modeled the service placement problem as a combinatorial auction market. The authors in [22] have given a reinforcement learning-based approach that manages the containers' horizontal and vertical elasticity. In the next step, container placement is performed by solving an integer linear programming problem or using a network-aware heuristic. Nevertheless, these works do not consider the dynamic migration of fog workloads across different fog nodes depending on the nodes' primary workload. Consequently, the micro-services execution performance suffers when there is a spike in the fog nodes' primary workload. In [32], the authors have proposed an dynamic programming-based offline micro-service coordination algorithm. Also, they have proposed a reinforcement learning-based online micro-service coordination algorithm for dynamic micro-service deployment. Table 1 presents the comparison of these works.

Our proposed approach attempts to address the limitations of the works mentioned above as PTC can provide micro-service isolation by using lightweight virtualization, i.e., containers. Again, PTC considers the dynamic workload of the fog computing environment by doing BO-based online learning to decide the best micro-service placement and a migration strategy. Therefore, in contrast to the existing works, we consider a fully dynamic system where the time-varying behavior of both the fog workloads and the computing nodes has been taken care of.

3 System architecture and design goals

The overall objective of the proposed PTC orchestration framework is to support the following four requirements for micro-service placement over a fog architecture—(a) development of a lightweight technology suitable for micro-service deployment over fog devices, (b) support service isolation for micro-service execution over fog devices, (c) design of an online micro-service placement mechanism catering to the dynamic workload and resource availability characteristics, and (d) seamless migration of micro-services to maintain application QoS of both the primary workload

Table 1 Comparison with existing approaches

Work	Containers	Micro-service	Migration	Bayesian optimization	Scalability
Stevant et al., 2018 [28]	✓	✓	✗	✗	✓
Saurez et al., 2016 [24]	✓	✓	✓	✗	✗
Elgamal et al., 2018 [7]	✓	✓	✗	✗	✓
Gonccalves et al., 2018 [10]	✗	✗	✓	✗	✗
Gu et al., 2015 [11]	✗	✗	✓	✗	✓
Li et al., 2019 [16]	✗	✓	✗	✗	✗
Yigitoglu et al., 2017 [34]	✓	✓	✗	✗	✓
Taneja et al., 2017 [31]	✗	✓	✗	✗	✓
Souza et al., 2018 [27]	✗	✓	✓	✗	✗
Yin et al., 2018 [35]	✓	✗	✓	✗	✗
Rossi et al., 2020 [23]	✓	✗	✗	✗	✓
Kayal et al., 2019 [13]	✗	✓	✗	✗	✓
Rossi et al., 2019 [22]	✓	✗	✗	✗	✓
Wang et al., 2019 [32]	✗	✓	✓	✗	✓
PTC	✓	✓	✓	✓	✓

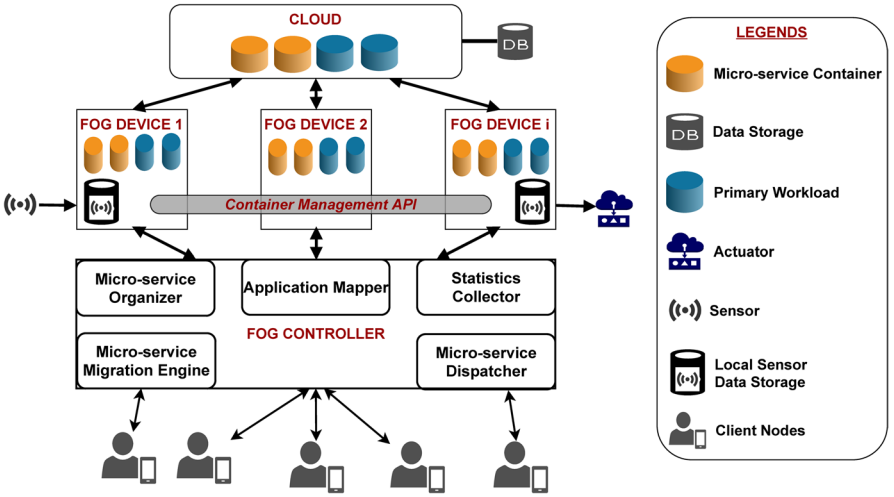


Fig. 1 PTC framework

and the micro-service workload. We discuss the architecture of the PTC framework (shown in Fig. 1) in this section.

Client Node The service request is generated by the client nodes. For example, a client might request the fog service to generate a consolidated report based on the sensing data captured by IoT devices. To do so, the client sends `REST` queries to the fog controller device.

Fog Controller Device The fog devices are connected to the fog controller device. This edge server receives the workload requests in terms of containerized micro-services and schedules them over the fog nodes. The different components of the *Fog Controller Device* are as follows. The `application mapper` maps the request–response to an application from where the requests originate. The `micro-service organizer` module divides the application into micro-services following the existing approaches such as ENTICE [14]. Also, the organizer module performs containerization of the micro-services. The `micro-service dispatcher` module finds the fog nodes to place the micro-services. The `statistics collector module` helps the dispatcher module to identify a suitable placement. The statistics collector module periodically monitors the available resources of the fog nodes. The micro-services are sent to the fog nodes by a `micro-service migration engine`, once the dispatcher module finds the placement. If the fog nodes become overloaded with their primary workloads, the micro-service dispatcher and the migration engines regenerate the placement.

Fog Device The containerized workloads are run in the fog devices. In Fig. 1, the blue-colored containers are the fog nodes' primary workloads. The micro-service containers utilize the remaining resources of the fog devices. We represent the micro-service containers with orange-colored containers (Fig. 1). The micro-service containers also need to communicate with the sensors and the actuators. The `local sensor data storage` module helps in this interaction. To seamlessly migrate the micro-service containers, we use a distributed `container management` API. The container management API is built using *"remote procedure call"* (`RPC`).

Cloud: The fog devices are connected to the cloud. Fog devices are always the first choice for placing a service. However, the proposed framework allows placement of a service in a cloud server if no fog server is available at a particular time instant.

In this framework, we combine two different approaches to design the overall execution model of PTC.

(a) We use containers as the sandboxing environment that supports the workload isolation along with the seamless migration of workloads across fog nodes.
(b) We design an online optimization strategy to decide the target fog node where a containerized micro-service can be placed. The optimization module facilitates both the initial micro-service deployment as well as the migration requirements. We use the standard application container-based approach [5] for executing the micro-services in the fog nodes and for migrating the micro-services based on the condition.

However, the challenge here is to design the dynamic micro-service placement in fog computing depending on the time-varying primary workload of the fog devices. The placement mechanism needs to be online, as well as it needs to cater to the dynamic workload characteristics of the system.

Next, we formally model the placement mechanism as an optimization problem. The details are given in Sect. 4.

4 Dynamic micro-service placement over fog nodes—an optimization formulation

This section presents the mathematical modeling of the fog micro-service placement problem and its theoretical analysis. The proposed execution model of the micro-services considers the following facts.

1. The fog devices have their primary workloads, which are time-varying, and thus can observe a spike in the resource demand. The priority is given to the primary workloads. Hence, the fog micro-service's execution might suffer if the primary workload withholds the resources used by the fog micro-services earlier.
2. To cope with such problems, PTC enables seamless migration of unfinished micro-services from one fog device to another. Such migrations are facilitated by container-based sandboxing. We consider that the migrations are live [20], indicating that execution states of the micro-services can be saved and later resumed on a different fog device.
3. The fog controller continuously monitors the available resource statistics on the fog devices and periodically decides the target fog device for a micro-service execution.

Considering the above mentioned facts, we present the system model for designing the optimization function for dynamic micro-service placement and execution over fog devices.

4.1 System model

We model the entire network, involving the end devices (sensors, actuators, edge devices), fog nodes (in-network devices like WiFi access point, gateway, router, core switches), fog controller, etc., as a weighted undirected communication graph. Table 2 shows the notations used. Let $G = (V,E,W)$ be an undirected and weighted communication graph. Here, the set of physical nodes is V, the set of physical communication links is E, and the set of edge weights is W. The sets of sensors, actuators and fog devices are denoted as $S = (S_i \in (1, \ldots, s))$, $\Lambda = (\Lambda_i \in (1, \ldots, \lambda))$ and $F = (F_i \in (1, \ldots, n))$, respectively. Here, the total number of sensors, actuators and fog devices are s, λ and n, respectively. Therefore, $V = \{S \cup \Lambda \cup F\}$. $e_{i,j} \in E$ represents the physical communication link between $v_i \in V$ and $v_j \in V$. In this case, the weight of each edge signifies the latency of the links.

We consider that the time is divided into l slots, where each slot is of length ι. Therefore, we define the system time vector $T = (\tau_t : t \in (1, \ldots, l))$. We also assume that each event either starts at the beginning of the slot or ends at the end of the slot. Therefore, all the time calculations done in this paper are in terms of number of slots. Let the weight of the shortest path between v_{i1} and v_{i2} be $D(v_{i1}, v_{i2})$, which signifies the communication latency between two nodes in the communication graph in terms of number of slots. Let k edge applications

Table 2 Notation table

Notation	Significance
G	An undirected and weighted communication graph
V	Set of physical nodes
E	Set of physical communication links
W	Set of edge weights
S	Set of sensors
Λ	Set of actuators
F	Set of fog nodes
s	Total number of sensors
λ	Total number of actuators
n	Total number of fog devices
$e_{i,j}$	The physical communication link between node v_i and node v_j
l	Number of time slots
ι	Length of a time slot
T	System time vector
$D(v_{i1}, v_{i2})$	Weight of the shortest path between node v_{i1} and node v_{i2}
k	Number of edge applications
$\mathbf{\Psi}$	Set of applications
$p_{x,y}$	y^{th} micro-service of a service P_x
h_x	Total number of atomic micro-services of a service P_x
$\mathbf{R}(F_i)$	The resource vector available in a fog node F_i
$\mathbf{\Gamma}(p_{x,y})$	The resource vector required by micro-service $p_{x,y}$
$\gamma_{x,y}^{q}$	The amount of resource type q required to execute $p_{x,y}$
f	The total different types of resources
r_i^q	The total available quantity of resource type q at F_i
A	Micro-service allocation matrix
$O(A, x, y)$	The occupancy of a fog node's resources by $p_{x,y}$ at a time instance
$Seq_{exe}(x, y)$	Fog device execution sequence by $p_{x,y}$
$[I]_{1\times n}$	The row vector of the fog device indices
T_{CPU}	Processing time
δ_i	Million instructions per second executed by fog node F_i
$\xi_{x,y}$	Million instructions required by $p_{x,y}$
$M_{x,y}$	Sub-vector of $Seq_{exe}(x, y)$
T_{MGR}	Migration time
$\Delta_{x,y}$	Constant delay factor for migration of $p_{x,y}$
ψ	The demand vector of sensors
α	The demand vector of actuators
$\psi_{x,y}$	The sensors required by $p_{x,y}$
α_x	The actuators required by $p_{x,y}$
$[I]_{1\times s}$	The row vector of the sensor indices
$[I]_{1\times \lambda}$	The row vector of the actuator indices
T_{DF}	The data fetch time
T_{ACT}	The actuation time

Table 2 (continued)

Notation	Significance
$\Omega(A, i, t)$	The executing micro-service vector at fog device F_i at time slot t
$\Theta(A, i, q, \Gamma, t)$	The amount of occupied resource of type q at time slot t
$T_{Resp}^{max}(A)$	The maximum response time taken by the applications
$\mathcal{R}(A)$	The resource availability at a particular time instant t
$\mathcal{D}_d$	The set of prior observations after d iterations
$\mu(\circ)$	The mean function
$K(\circ, \circ)$	The covariance kernel function
Φ	The standard normal cumulative distribution function
ϕ	The standard normal density function
EI	Acquisition function
ζ	Noise

are present in the system, such that $\mathbf{\Psi} = \left(P_x : x \in (1, \ldots, k)\right)$. $\mathbf{\Psi}$ is the set of applications. These applications need to execute their micro-services over the fog devices. An application can be decomposed into multiple micro-services, as mentioned earlier. We define $p_{x,y}$ as the y^{th} micro-service of P_x. Hence, $P_x = \left(p_{x,y} : y \in (1, \ldots, h_x)\right)$, where h_x is the total number of atomic micro-services of P_x. We consider that each micro-service can execute independently.

We denote $\mathbf{R}(F_i)$ as the resource vector available in a fog node F_i. This resource vector is time-varying and keeps on changing depending on the primary workload of the fog device. The resource vector required by micro-service $p_{x,y}$ is denoted by $\mathbf{\Gamma}(p_{x,y})$. We assume that the initial resource vector required by $p_{x,y}$ does not change over time, however depending on the partial completion of a micro-service's execution, the residual resource requirement might change. $\mathbf{\Gamma}(p_{x,y}) = (\gamma_{x,y}^q \in \mathbb{R} : q \in (1, \ldots, f))$, where each component $\gamma_{x,y}^q$ signifies the amount of resource type (central processing unit, memory, network bandwidth, etc.) q required to execute $p_{x,y}$ and f is the total different types of resources. Similarly, $\mathbf{R}(F_i) = (r_i^q \in \mathbb{R} : q \in (1, \ldots, f))$, where r_i^q is the total available quantity of resource type q at F_i. Each fog device F_i can host more than one micro-services depending on its available resources.

We define a micro-service allocation matrix $[A] = (A_{x,y,i,t} \in (0, 1) : (x \in (1, \ldots, k), y \in (1, \ldots, h_{max}), i \in (1, \ldots, n), t \in (1, \ldots, l))$ that gives the mapping between micro-services and fog nodes at each time stamp, where $h_{max} = \max_x(h_x)$, and each element $A_{x,y,i,t} = 1$ if $p_{x,y}$ is assigned to F_i at time slot t, otherwise $A_{x,y,i,t} = 0$. The task of the fog controller is to generate this occupancy metric dynamically based on its observation of the available resources at the fog devices. The allocation matrix changes over time, and such changes typically trigger a service migration from one fog device to another. Consequently, the occupancy ($O(A, x, y)$) of a fog node's resources by $p_{x,y}$ at a time instance is defined as follows.

$$\left[O(A,x,y)\right]_{n\times l} = \left(A_{x,y,i,t} \in (0,1) \,:\, i \in (1\ldots n), t \in (1\ldots l)\right) \tag{1}$$

Due to the dynamic workload characteristics of the fog devices, we consider that a micro-service may complete its execution over multiple fog nodes. Therefore, the time to calculate the total response time for an application has the following components—(a) processing time, (b) migration time and (c) data fetch and actuating time. We compute these three components as follows.

4.2 Computation of the processing time

As we mentioned earlier, a micro-service may get placed over a set of fog devices sequentially to complete its execution. Let the fog device execution sequence by $p_{x,y}$ be $Seq_{exe}(x,y) = [I]_{1\times n} \times [O(A,x,y)]_{n\times l}$, where $[I]_{1\times n} = [1,2,\ldots,n]$ represents the row vector of the fog device indices. We give the processing time (T_{CPU}) required by $p_{x,y}$ in order of number of time slots as follows. Here, F_i is capable of executing δ_i million instructions per second (MIPS), and $p_{x,y}$ requires $\xi_{x,y}$ million instructions.

$$T_{CPU}(x,y,A) = \sum_{i \in Seq_{exe}(x,y)} \frac{\xi_{x,y}}{\delta_i t} \tag{2}$$

4.3 Computation of the migration time

During the execution, a micro-service may require migration from one fog device to another, which incurs additional migration time. Let the sub-vector of $Seq_{exe}(x,y)$ be $M_{x,y} = (M^b_{x,y} = Seq_{exe}(x,y)_b \,:\, Seq_{exe}(x,y)_b \neq Seq_{exe}(x,y)_{b+1})$, where b is a particular time slot. Here, the fog node migration vector by $p_{x,y}$ is $M_{x,y}$. Therefore, the migration time (T_{MGR}) is computed as follows.

$$T_{MGR}(x,y,A) = \sum_{M^b_{x,y}} (\Delta_{x,y} \times D(M^b_{x,y}, M^{(b+1)}_{x,y})) \tag{3}$$

Here, $\Delta_{x,y}$ is a constant delay factor for migration of $p_{x,y}$, which depends upon the amount of data to be transferred from one fog device to another. Also, the weight of the shortest path between v_i and v_j is $D(v_i, v_j)$. $D(v_i, v_j)$ is signifying the communication latency between two nodes in terms of number of slots.

4.4 Computation of the data fetch and actuating time

Apart from migration, communication overhead plays a significant role in initial data fetching from sensors and triggering actuators. Let the demand vector of sensors and actuators are $\psi = (\psi_{x,y,i} \,:\, x \in (1,\ldots k), y \in (1,\ldots h_{max}), i \in (1,\ldots s))$ and $\alpha = (\alpha_{x,i} \,:\, x \in (1,\ldots k), i \in (1,\ldots \lambda))$, respectively. Here, $\psi_{x,y,i} = 1$ if $p_{x,y}$ requires S_i, otherwise it is 0. Similarly, $\alpha_{x,i} = 1$ if P_x requires Λ_i and it is 0 for all other cases. Let $\psi_{x,y} = [I]_{1\times s} \times (\psi_{x,y,i} \,:\, i \in (1,\ldots s))$ and $\alpha_x = [I]_{1\times\lambda} \times (\alpha_{x,i} \,:\, i \in (1,\ldots \lambda))$

are the sensors and actuators required by $p_{x,y}$, respectively. The row vector of the sensor indices is $[I]_{1\times s} = [1, 2, \ldots, s]$ and the row vector of the actuator indices is $[I]_{1\times\lambda} = [1, 2, \ldots, \lambda]$. The data fetch time ($T_{DF}$) of $p_{x,y}$ is given below.

$$T_{DF}(x, y, A) = \max_{i\in\psi_{x,y}}(D(M^1_{x,y}, i)) \tag{4}$$

The actuation time (T_{ACT}) is given below.

$$T_{ACT}(x, y, A) = \max_{i\in\alpha_x}(D(M^{|M_{x,y}|}_{x,y}, i)) \tag{5}$$

We define the executing micro-service vector ($\Omega(A, i, t)$) at fog device F_i at time slot t as follows.

$$\Omega(A, x, y, i, t) = \left(A_{x,y,i,t} \in (0, 1) : x \in (1 \ldots k), y \in (1 \ldots h_{max})\right) \tag{6}$$

The amount of occupied resource of type q at time slot t is given as follows.

$$\Theta(A, i, q, \mathbf{\Gamma}, t) = \left(\sum_{x,y}\left(\Omega(A, x, y, i, t) \times \gamma^q_{x,y}\right)\right) \tag{7}$$

According to the capacity property, cumulative occupied resources by micro-services executing on a single fog device must not surpass the total available resource at that fog device. A valid allocation matrix must satisfy the following capacity property given in Equation (8).

$$\underset{i,q,t}{\forall}\ \Theta(A, i, q, \mathbf{\Gamma}, t) \leq r^q_i \tag{8}$$

If the capacity property is satisfied by the allocation matrix, then the total response time required by $\mathbf{\Psi}_x$ can be calculated using Equations (2) to (5). Therefore, we calculate the total response time required by $\mathbf{\Psi}_x$ using Equations (2) to (5), as follows.

$$T_{Resp}(A, x) = \max_{y\in(1\ldots h_{max})}\begin{pmatrix} T_{DF}(x, y, A) + T_{CPU}(x, y, A) \\ +T_{MGR}(x, y, A) + T_{ACT}(x, y, A) \end{pmatrix} \tag{9}$$

4.5 Problem definition

We represent the formal definition of the micro-service placement problem as follows. Given the communication graph G and available resources ($\mathbf{R}(F_i)$), the micro-service placement problem finds an allocation schedule (A) for each micro-service ($p_{x,y}$) with required instructions ($\xi_{x,y}$) and resources ($\mathbf{\Gamma}(p_{x,y})$) such that the maximum response time taken by the applications ($T^{max}_{Resp}(A)$) is minimized. Therefore, mathematically, the problem can be represented as an optimization problem given by Equation (10). Therefore, we have

$$\begin{aligned} \underset{A}{\text{minimize}} \quad & T^{max}_{Resp}(A) \\ \text{subject to:} \quad & \\ & \mathcal{R}(A) \geq \mathbf{Z} \end{aligned} \tag{10}$$

where $\mathcal{R}(A)$ is the resource availability at a particular time instant t and $\mathbf{Z} = \left(z_{i,q,t} = 0 : i = (0, \ldots, n), q = (0, \ldots, f), t = (0, \ldots, l)\right)$.

In *"Minimax facility location problem"* (MFLP) [6], the cost to transfer the items to the demand site is optimized. In our micro-service allocation problem, we are interested to minimize the cost, i.e., the response time. We now show that the *"Minimax facility location problem"* (MFLP) [6] is polynomial time reducible to micro-service allocation problem (Equation (10)). Thus, the micro-service allocation is $\mathcal{NP}$-hard.

Theorem 4.1 *"Minimax facility location problem"* (*MFLP*) [6] *is polynomial time reducible to micro-service allocation problem given in Equation* (10).

Proof Let $\{L_i : i \in (1, \ldots, n)\}$ be the set of locations where a warehouse can be placed and $\{C_j : j \in (1, \ldots, m)\}$ be the set of demand sites that must be serviced. Suppose $c_{i,j}$ be the cost of delivery of items from L_i to C_j. In that case, the MFLP finds the subset of locations where the warehouses should be opened such that the maximum cost to transfer the items from facilities to the demand site is minimized. This is a known $\mathcal{NP}$-hard problem. To prove the $\mathcal{NP}$-hardness of our micro-service placement, we encode the instance mentioned above of MFLP.

The encoding of the instance mentioned above of MFLP can be done in the following way. In our case, the system consists of m applications, each having single micro-services (μ_j). Each of the micro-services (μ_j) needs to be placed in exactly one of the fog devices. Let $\{L_i : i \in (1, \ldots, n)\}$ be the set of fog devices and $\{C_j : j \in (1, \ldots, m)\}$ be the set of micro-services to be placed. $c_{i,j}$ is the cost (in terms of response time) of placing the j^{th} micro-service (C_j) in i^{th} fog device (L_i). The reduction can be made in polynomial time. This completes the polynomial time encoding of MFLP to our micro-service placement. The used fog devices represent the location where the warehouses can be set up from the solution allocation matrix. However, as the MFLP is a known $\mathcal{NP}$-hard problem; therefore, we can claim that our proposed micro-service placement problem is $\mathcal{NP}$-hard. □

5 Solution approach: Bayesian optimization for micro-service placement

As the optimization problem is *NP*-hard, we design an approximate solution of the proposed optimization, which is fast and provides quick decision about the placement depending on the system's dynamicity. We use Bayesian optimization (BO), a reinforcement learning framework that provides the solution of an optimization based on the historical observation and posterior distribution of the solution vector.

BO performs well in the scenarios when conducting a single experiment takes a higher time.

We define the micro-service allocation matrix as a solution configuration of BO. BO optimizes the objective function based on the prior observations and posterior distribution by conducting iterative experiments over the solution configurations. In our case, conducting an experiment is equivalent to test the performance of a given allocation matrix which is costly in terms of time taken to perform a test. Surprisingly, BO performs well in such cases where performing one single experiment takes a higher time.

To formulate the BO framework, we assume that the utility function $T^{max}_{Resp}(A)$ follows a normal distribution. BO executes initial experiments based on the prior belief function. After sufficient number of experiments, BO modifies the prior belief function based on the posterior distributions. BO uses an *"acquisition function"* (*EI*) to pick the configurations for the iterations. This leads to a near-optimal solution. Let $\mathcal{D}_d = \{(a_1, T^{max}_{Resp}(a_1)), \ldots, (a_d, T^{max}_{Resp}(a_d))\}$ be the set of prior observations after d iterations. We denote $p(T^{max}_{Resp}) = \mathcal{N}(\mu, K)$. Here, the mean function is $\mu(\circ)$, and the covariance kernel function is $K(\circ, \circ)$. We define the mean and covariance kernel function as follows.

$$\mu(a_u) = \mathbb{E}(T^{max}_{Resp}(a_u)) \tag{11}$$

$$K(a_u, a_v) = \mathbb{E}\Big(\Big(T^{max}_{Resp}(a_u) - \mu(a_u)\Big)\Big(T^{max}_{Resp}(a_v) - \mu(a_v)\Big)\Big) \tag{12}$$

Let us denote Φ and ϕ as the standard normal cumulative distribution function and the standard normal density function, respectively. We define $U_{min} = \underset{a \in \mathcal{D}_d}{min}(T^{max}_{Resp}(a))$, $\pi = \frac{U_{min} - \mu(a_d)}{\sigma(a_{d-1}, a_d)}$, and $\sigma(a_{d-1}, a_d) = \sqrt{K(a_{d-1}, a_d)}$. We choose the acquisition function (Equation (13)) which is recommended in [3].

$$EI(A|\mathcal{D}_d) = \begin{cases} 0 & \text{if:} \sigma(a_{d-1}, a_d) = 0 \\ \Big(\big(U_{min} - \mu(a_d)\big)\Phi(\pi)\Big) + \big(\phi(\pi)\sigma(a_{d-1}, a_d)\big) & \text{Otherwise} \end{cases} \tag{13}$$

However, $EI(A|\mathcal{D}_d)$ works well in case of unconstrained optimization. We take the procedure suggested by Gardner et al.[8] to satisfy our constrained optimization. By following their approach, we assume that $\mathcal{R} \Leftarrow \mathcal{A} \Rightarrow$ follows Bernoulli process, and *EI* can be modified to Equation (14).

$$EI^c(A|\mathcal{D}_d) = P(\mathcal{R}(A))EI(A|\mathcal{D}_d) \tag{14}$$

There can be observation noise in our setup due to a rise in loads in the fog devices, network delay, etc. Therefore, the proposed BO algorithm must handle noise. We assume that the noise (ζ) is a normally distributed random variable with zero mean, i.e., $\zeta = \mathcal{N}(0, \sigma_\zeta)$.

Algorithm 1: PTC

Input: Resource avalibility of the fog nodes, Resource requirement of the applications
Output: Allocation matrix for which the maximum response time taken by the applications is minimized

1 **Function** *Main()*:
  /* Function for Bayesian Optimization */
2 $\mathcal{D}_0 \leftarrow \emptyset$;
3 **for** $(d \leftarrow 1; HasConverged(\mathcal{D}_{(d-1)}); d++)$ **do**
4 $a_{nxt} \leftarrow \emptyset$;
5 **while** $Length(a_{nxt}) < n_{conf}$ **do**
6 Choose an allocation matrix a_i such that $a_i \notin \{\mathcal{D}_{(d-1)}\}$;
7 **if** *IsFeasible*(a_i) **then**
8 $a_{nxt} \leftarrow \{a_{nxt} \cup a_i\}$;
9 **else**
10 **go to** 5;
11 a_d = Configuration for which the aquisition function is minimum;
12 $\mathcal{D}_d = \left\{\mathcal{D}_{(d-1)} \cup \left(a_d, T^{max}_{Resp}(a_d) + \zeta\right)\right\}$;
13 **return** allocation matrix from $\mathcal{D}_d$ for which the maximum response time taken by the applications is minimized;

1 **Function** *IsFeasible*(a_κ):
  /* Checks feasibility of the allocation matrix */
2 **for** *all applications* $P_x \in \Psi$ **do**
3 **for** $p_{x,y} \in P_x$ **do**
4 $c = \{1, 2, \ldots f\}$;
5 **if** $\nexists_{i \in c} : r^q_i > \gamma^q_{x,y}$ **then**
6 **return** *False*;
7 **else**
8 Remove i from c;
9 **return** *True*;

PTC is described in Algorithm 1. In the Main() function, $\mathcal{D}_0$ is initialized to empty set in Line 2. Lines 3-12 iteratively finds the allocation or configuration for which the aquisition function is minimum. a_{nxt} is initialized to empty set in Line 4. Lines 5-10 generates $(n_{conf} - 1)$ numbers of allocation matrices (a_i). Each of the generated allocation matrices is checked for feasibility. If the allocation matrix is feasible, then that configuration is tested by applying it to the system. Once the testing process is over, the objective function is evaluated, and the next configuration (a_d) is determined based on the minimization of the acquisition function. Line 11 finds the configuration (a_d) for which the aquisition function is minimum. In Line 12, this new configuration (a_d) is added to the set of prior observations set, i.e., $\mathcal{D}_d$ set. This process is repeated until the system converges. PTC algorithm runs iteratively and it stops after reaching convergence. In the IsFeasible(a_κ) function, Line 2 runs for all applications. Line 3 runs for all micro-services of an application. In Line 4, c is the set of all resource types. Line 5-8 checks if any fog node is able to satisfy all the resource requirements of a micro-service. In each iteration, the PTC algorithm checks resource constraint for every micro-service. Therefore, the

computational complexity of PTC is $\mathcal{O}(k * h_x)$. Here, k is the number of edge services and h_x is the total number of micro-services of a service.

6 Performance evaluation

We have implemented the PTC framework in a testbed and a public cloud (Amazon EC2). In the testbed, we analyze the performance of PTC in a realistic environment under a constrained setup. To analyze the system scalability of PTC, we perform the experiments on a large scale over Amazon EC2.

6.1 Testbed setup

In the testbed, the fog nodes are implemented using `Raspberry Pi 3 model b` (https://www.raspberrypi.org/products/raspberry-pi-3-model-b/ (Access: 2021/10/13 08:51:59)) devices, which works like a network gateway. The fog nodes are connected over a local network and provide packet forwarding services to the connected devices. The devices have been configured with `Ubuntu 16.04.2 long-term support (LTS)` (https://www.ubuntu.com/) operating system. `Docker` (https://www.docker.com/) containers are used to provide the sandboxing environment of the micro-services running over the fog nodes. Docker provides a platform for lightweight containers by application layer virtualization. The PTC framework is connected to a private cloud available in our institute.

Figure 2 shows the entire topology of the testbed architecture, which emulates the framework components, as shown in Fig. 1. We give the description of the components used in testbed experiments in Table 3. An ethernet switch has been used to connect various networking components to the local area network (LAN) in the testbed. We have used two switches to connect the fog devices and the client nodes to the ethernet switch. These switches interconnect various fog devices, the fog controller and the client node. A virtual machine running in the private institute cloud is used to host micro-services using Docker over a cloud environment. The fog controller device has `Ubuntu 16.04.2 LTS` (https://www.ubuntu.com/) as the operating system. Client applications are executed on another workstation having a similar configuration to that of the fog controller device. However, the client node only uses a custom TCP socket program to request the fog applications.

6.2 Experimental methodology

Table 4 provides the details of the values taken during the experiments in the testbed. The delays have been set up using the Unix `tc` utility (http://manpages.ubuntu.com/manpages/xenial/man8/tc.8.html). We use `stress-ng` (http://manpages.ubuntu.com/manpages/artful/man1/stress-ng.1.html) for emulation of memory load due to the primary workload in the fog devices. For memory status monitoring of fog devices and the fog controller, we use Linux `free utility` (http://manpages.ubuntu.com/manpages/bionic/man1/free.1.html). The CPU usage is measured

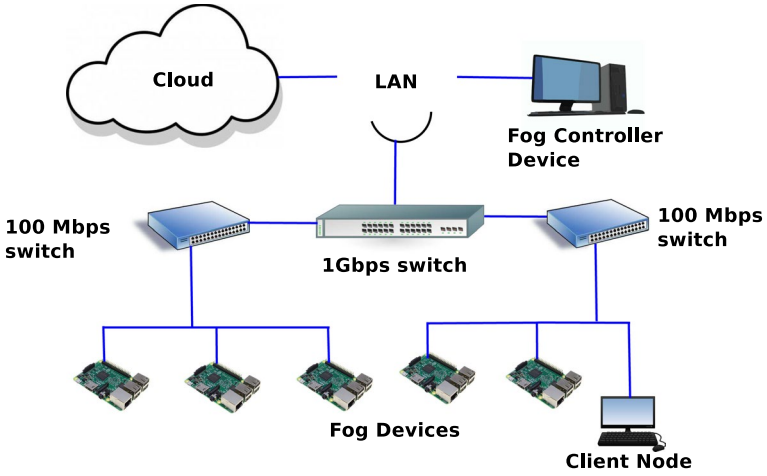


Fig. 2 Testbed topology for the experiments

Table 3 Devices used in testbed experiments

Component	Description
Fog devices	`Raspberry Pi 3 model b` devices (a quad-core 1.2 GHz Broadcom BCM2837 64-bit central processing unit (CPU) and 1 GB random access memory (RAM))
Ethernet wwitch	Netgear GS608 8-Port 1 Gigabit ethernet switch
Desktop wwitches	TP-Link TL-SF1008D 8-Port 10/100Mbps switches
Cloud VM	A cloud VM (4 GB RAM with 2 CPUs) running in the private institute cloud
Fog controller device	Quad-core 3.2 GHz Intel Core i5-4570 processor with 4 GB of RAM
Client node	Quad-core 3.2 GHz Intel Core i5-4570 processor with 4 GB of RAM

using the `iostat` utility (https://man7.org/linux/man-pages/man1/iostat.1.html). The network status is obtained by the `bmon` utility (http://manpages.ubuntu.com/manpages/bionic/man8/bmon.8.html). We use `scp` (http://manpages.ubuntu.com/manpages/trusty/man1/scp.1.html) to transfer files from the fog devices to the fog controller. `Scikit-Optimize (skopt)` (https://scikit-optimize.github.io/) python library is used for the implementation of the BO algorithm. We consider `optical character recognition (OCR)` as a service for testing; for this, we have run `tesseract-OCR web service` (https://hub.docker.com/r/guitarmind/tesseract-web-service/) as a docker container. Also, we have tested PTC with an NLP application, as discussed in Sect. 6.8.

6.3 Competing heuristics

We consider *Foglets* [24], *first-fit* [27], *best-fit* [27] and *mobility-based* [10] mechanisms as the baseline methods to analyze the performance of PTC. In Foglets, the

Table 4 Values of different parameters used in testbed

Parameter	Value
Number of fog devices	5
Number of application types	1
Number of micro-services per service or application	5
Network delay: RTT for fog device–fog device	10–18 ms (if not closest)/0.5 ms (if closest)
Network delay: RTT for fog device–cloud	200 ms
Network delay: RTT for controller device–cloud	200 ms
RAM load in fog devices	20 MB (low loaded)/100 MB (high loaded)
RAM needed for docker container, i.e., micro-service	40 MB
Number of parallel applications or number of scenarios	1 to 8
Number of runs per scenario	3-4

authors have proposed a programming infrastructure for geo-distributed applications, which uses a discovery and deployment protocol to find the fog computing devices with sufficient resources to host an application component. The first-fit mechanism selects the available fog devices for deployment and processing of an image. On the contrary, the best-fit method sorts the processing requests in ascending order according to their demands. In the mobility-based mechanism, the authors have developed a placement to minimize the applications' latency.

6.4 Application response time

We show the performance of PTC in terms of response time in Fig. 3(a). It is observed that PTC is faster than the baselines. The distribution of average response time for PTC converges within 50 seconds. At the same time, the distribution of average response time for baselines converges between 53 seconds and 88 seconds. It is found that PTC converges quickly to an optimal or near-optimal value (within 30 iterations). Table 5 presents a performance comparison of PTC with respect to the baseline approaches as shown in Fig. 3(a).

The PTC algorithm uses BO-based reinforcement learning. As BO can reach the desired solution in fewer iterations, the response time of the application reduces. BO finds the selected points in the search space with relatively few function evaluations. This optimization technique induces adaptive learning over the PTC framework and therefore makes PTC intelligent in dynamically selecting the target fog devices for micro-service placements. The BO-based adaptive reinforcement learning framework helps PTC optimize the utility function with prior observations and posterior distribution, such that PTC can reduce the overall response time. Consequently, the proposed framework learns the dynamic fog environment and their primary

Table 5 Performance of PTC in terms of average response time

Approach	Average response time
Foglets [24]	24.21–80.35 sec
First-fit [27]	16.74–88 sec
Best-fit [27]	11.48–57.39 sec
Mobility-based [10]	12–53 sec
PTC	9.71–50 sec

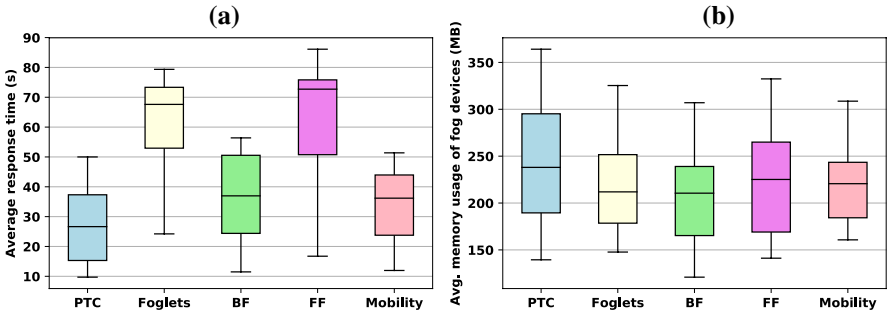


Fig. 3 **a** Distribution of average response time and **b** Distribution of average memory usage of the fog devices

workloads over time and can decide accordingly to support the applications' low response time.

On the contrary, the Foglets mechanism places the application components in fog devices that are closest. Therefore, it cannot find a suitable allocation matrix as the nearest fog nodes' computing resources might be blocked with their primary workloads. In the first-fit mechanism, the available fog devices which have sufficient resources are chosen for the placement. However, this method also does not consider the dynamic primary workload. Thus, in this case, the response time is more than PTC. The best-fit algorithm places the micro-services according to their resource demand. The best-fit response time is also more than PTC as the best-fit algorithm takes more time to deploy the micro-services in fog devices.

On the other hand, mobility-based placement only checks if the applications' latency is minimized while allocating them in the fog devices. This approach tries to maximize the number of tasks deployed in the fog landscape, ensuring the latency constraint. Thus, the average response time is higher in the mobility-based scheme than PTC, particularly when the primary workload observes a spike in the resource demand.

6.5 Resource usage of the fog devices

The memory consumption of the fog nodes is depicted in the Fig. 3b. It may be observed that the PTC consumes more amount of memory than the baseline

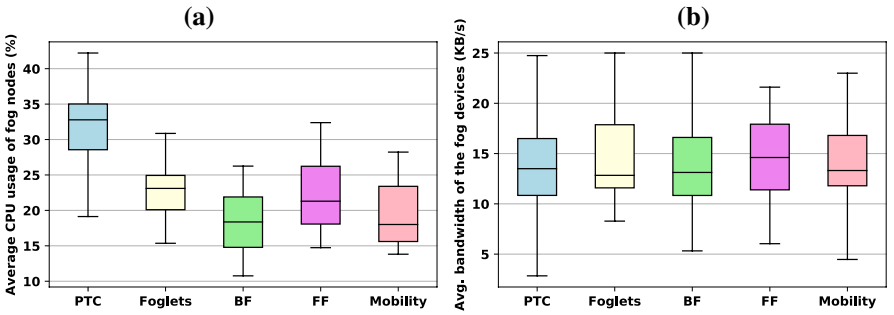


Fig. 4 **a** Distribution of average CPU usage of the fog devices and **b** Distribution of average network bandwidth usage of the fog devices

algorithms. The density of memory usage distribution is higher in PTC (in the range of 139 MB–366 MB) than the baselines. Further, we show the average CPU usage distribution of the fog nodes in Fig. 4a. The CPU usage is also higher in PTC (in the range of 18.43%–44%) than the CPU usage of the baseline algorithms. In a similar line, the bandwidth usage is also higher in PTC (in the range of 3 KB/s–24.74 KB/s) in comparison with other baselines, as shown in the Fig. 4b.

As these statistics are collected over the fog nodes, it indicates the average resource consumption by the micro-services placed on that fog nodes. The figures suggest that PTC can provide better resource utilization to the running micro-services compared to other baselines. We observe that the baseline mechanisms fail to place the micro-services in the most optimized fog devices having better availability of excess resources over time. They only consider the instantaneous measurements and, therefore, do not consider the fog devices' time-varying primary workload characteristics. Consequently, the running micro-services get better computing platform in PTC, resulting in a low response time, as we have seen earlier.

6.6 Resource usage of the fog controller

We show the resource consumption of the fog controller node in Fig. 5(a). As the proposed PTC algorithm runs for multiple iterations, the PTC algorithm's memory consumption is more than the baselines. The controller's average memory usage is higher (in the range of 756 MB–922 MB) in PTC than the baselines. The average CPU consumption of the fog controller device is shown in Fig. 5(b). It is also higher in PTC (in the range of 26%–74%). The average bandwidth usage of PTC is shown in Fig. 6(a). It is also higher (in the range of 11 KB/s–87.28 KB/s) in PTC compared to the baselines. In summary, we observe that the trade-off is in the fog controller's resource usage, where the PTC controller needs more resources than other baselines. However, considering the improvement observed in the application response time, this trade-off is acceptable as the average increase in the resource usage is not very significant, although it is higher than the baselines.

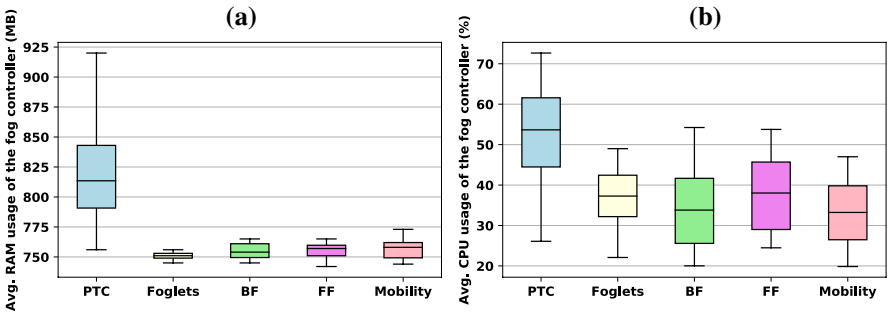


Fig. 5 **a** Distribution of average memory usage of the fog controller device and **b** Distribution of average CPU usage of the fog controller device

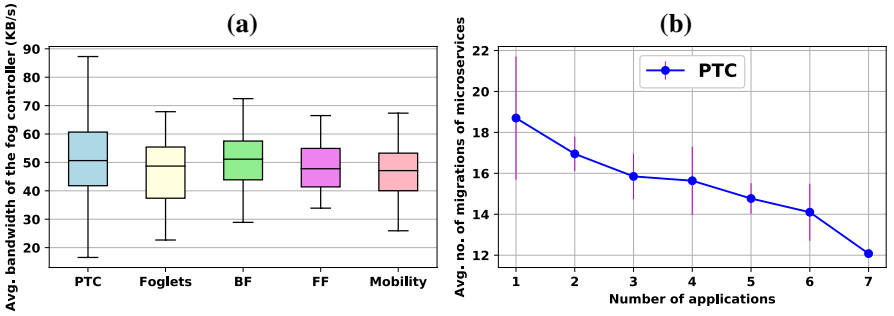


Fig. 6 **a** Distribution of average network bandwidth usage of the fog controller device and (b) Avg. number of migrations of the micro-services

6.7 Average number of micro-service migrations

The proposed PTC algorithm migrates the containerized micro-services to a new fog node if the current fog node becomes loaded. The migration ensures the minimization of service response time. The average number of migrations of the micro-services in Fig. 6b. It is observed that though the total number of migrations increases, the average number of migrations decreases with the increase in the number of applications. It indicates the system's stability at high load.

6.8 Case study with an NLP (Natural Language Processing) application

We have tested PTC with an NLP application—text summarization toolbox[1]. Such an application is important because it supports many practical situations. The evaluation is performed in our fog computing testbed with the configurations given in

[1] https://glowingpython.blogspot.com/2014/09/text-summarization-with-nltk.html (Access: 2021/10/13 08:51:59).

Table 6 Performance of PTC in terms of average response time for the NLP application

Approach	Average response time
Foglets [24]	59.12–131 sec
First-fit [27]	58.86–142.2 sec
Best-fit [27]	49.59–144 sec
Mobility-based [10]	49.37–143.29 sec
PTC	36.75–107 sec

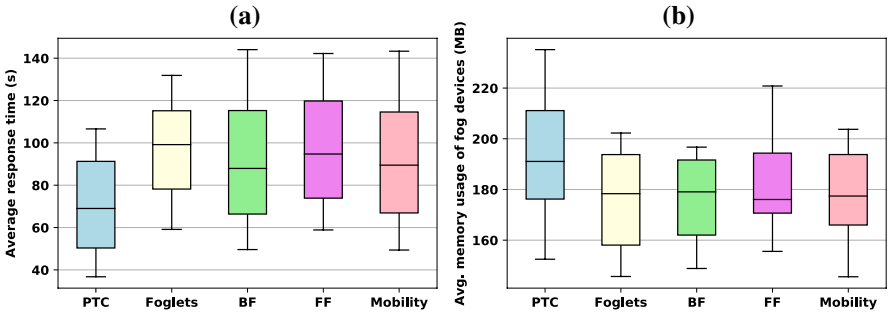


Fig. 7 **a** Distribution of average response time and **b** Distribution of average memory usage of the fog devices

Table 4. The testbed setup given in Sect. 6.1 has been used for the case study. It is observed that the average response time is less for PTC than the baselines. The distribution of the response time in PTC converges within 107 seconds (Fig. 7a), whereas distributions of the response time for the baseline algorithms converge between 131 seconds and 144 seconds. Table 6 shows a performance comparison of PTC with respect to the baseline approaches for the NLP application as shown in Fig. 7a.

The average memory usage is more in PTC, as we have seen earlier. It is in the range of 152 MB–235 MB (shown in Fig. 7b), whereas the average memory distribution of baselines converges within 220 MB for first-fit, within 202 MB for Foglets, within 203 MB for mobility-based and within 196 MB for best-fit, respectively. The CPU usage of the fog devices is higher in PTC in the case of the NLP application. It is in the range of 6%–19% (Fig. 8a) for PTC. The CPU usage distribution of baselines converges within 13% for first-fit, within 14% for best-fit and Foglets methods. The distribution of CPU usage converges within 15% for mobility-based algorithm. The distribution of bandwidth usage of the fog devices is also marginally higher in PTC than the baseline algorithms. For PTC, it is in the range of 5 KB/s–17 KB/s (Fig. 8b). On the contrary, the distribution of network bandwidth usage converges within 18 KB/s for the best-fit mechanism, within 17 KB/s for Foglets and within 16 KB/s for the first-fit and the mobility-based algorithm.

Similar to the previous observation, the average memory usage distribution of the controller is more in PTC compared to other baselines, which is in the range of 756 MB–930 MB (Fig. 9a). The fog controller's average memory usage distribution

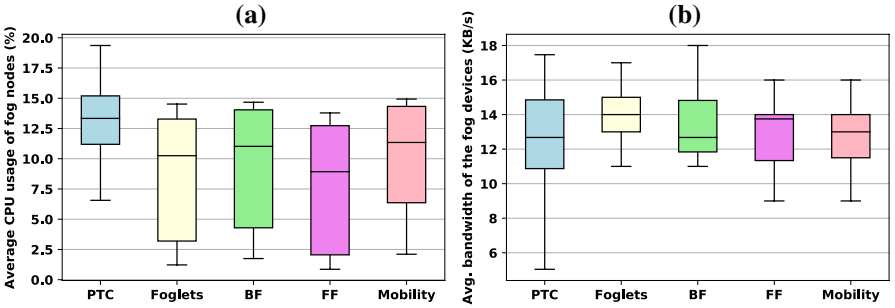


Fig. 8 **a** Distribution of average CPU usage of the fog devices and **b** Distribution of average network bandwidth usage of the fog devices

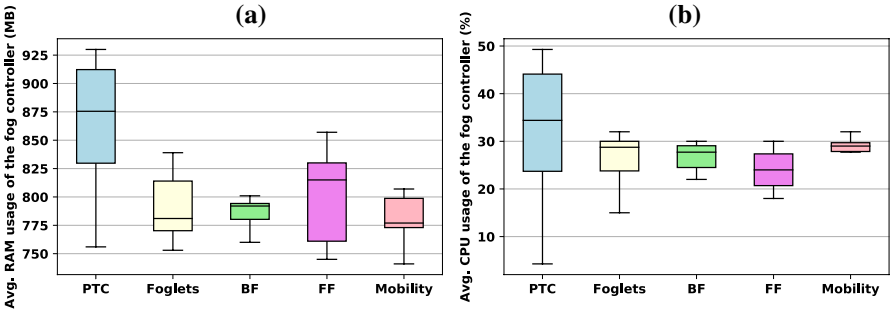


Fig. 9 **a** Distribution of average memory usage of the fog controller device and **b** Distribution of average CPU usage of the fog controller device

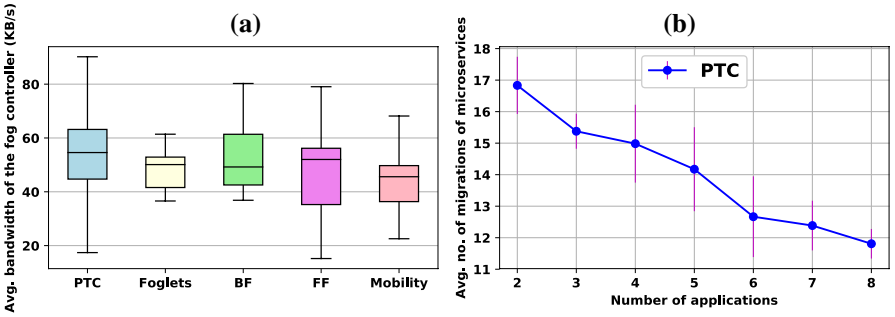


Fig. 10 **a** Distribution of average network bandwidth usage of the fog controller device and **b** Avg. number of migrations of the micro-services

converges within 839 MB for Foglets, within 800 MB for best-fit, within 857 MB for first-fit and within 807 MB for the mobility-based algorithm. The CPU usage of the controller is in the range of 4–49% for PTC. The CPU usage is shown in Fig. 9b. The distribution of baselines converges within 35% for mobility-based algorithm and 32% Foglets, respectively. It converges within 30% for best-fit and first-fit

Table 7 Performance of PTC in terms of average response time for scalability analysis

Approach	Average response time
Foglets [24]	175–650.82 sec
First-fit [27]	211.66–755.34 sec
Best-fit [27]	196–775 sec
Mobility-based [10]	214–634 sec
PTC	106.6–w318 sec

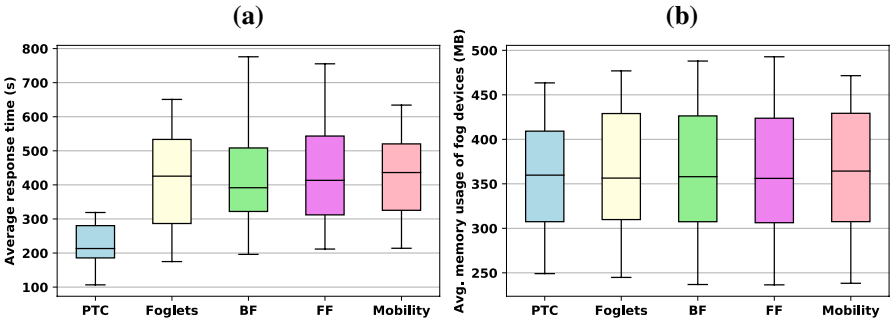


Fig. 11 **a** Distribution of average response time and **b** Distribution of average memory usage of the fog devices

algorithms. The distribution of the average network bandwidth usage of the PTC controller is marginally higher than other baselines. It is in the range of 17 KB/s–98 KB/s (Fig. 10a) for PTC. The distributions of baselines converge within 80 KB/s for best-fit, within 79 KB/s for first-fit, within 61 KB/s for Foglets and within 68 KB/s for the mobility-based mechanism.

The average number of migrations of the micro-services for the NLP application is shown in Fig. 10b. In this case, we also observe that though the total number of migrations increases, the average number of migrations drops with the increase in the number of applications, indicating the system's stability.

6.9 Scalability analysis in Amazon EC2

To test the scalability of the system, we have experimented with Amazon EC2. We have taken `tesseract-OCR web service` as a docker container for testing. We have taken 45 virtual machines (VM) in Amazon EC2. These VMs are considered as the fog devices for the experiments. The VMs have a 2.5 GHz Intel Xeon Family Processor with 1 GB RAM. The number of containers is increased from 80 to 360. The fog controller device has 1 GB of RAM. We show the average response time in 11a. We find that our proposed PTC framework can decrease the services' average response time than the baselines. The distribution of average response time for PTC converges within 318 seconds (Fig. 11a). Distributions of the baselines converge between 634 seconds and 775 seconds. On

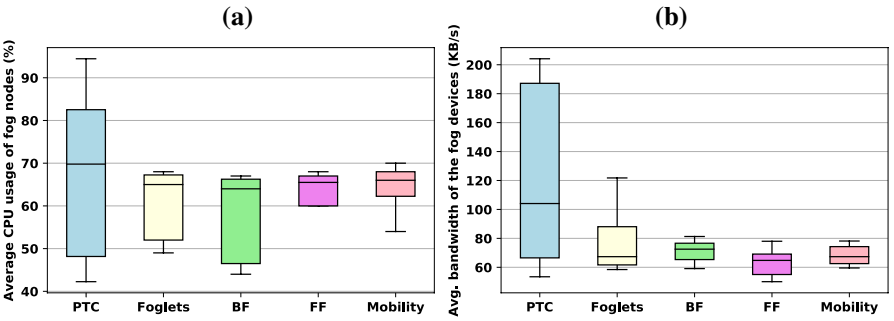


Fig. 12 **a** Distribution of average CPU usage of the fog devices and **b** Distribution of average network bandwidth usage of the fog devices

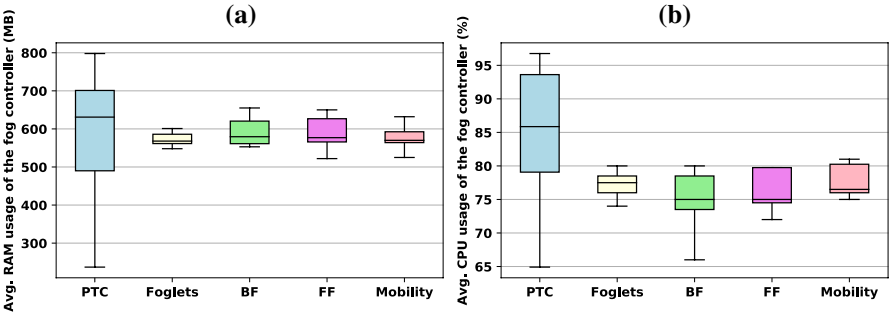


Fig. 13 **a** Distribution of average memory usage of the fog controller device and **b** Distribution of average CPU usage of the fog controller device

average, PTC performs better than the baselines in Amazon EC2 experiments. Thus, our PTC framework is scalable. Table 7 shows a performance comparison of PTC with respect to the baseline approaches for the scalability analysis as shown in Fig. 11a.

It may be observed that the average memory usage of the fog devices is in the range of 250 MB–463 MB in PTC in Amazon EC2 (shown in Fig. 11b). The average memory usage distribution for baseline algorithms converges within 492 MB for first-fit, within 476 MB for Foglets, within 471 MB for mobility-based and within 487 MB for the best-fit algorithm, respectively. The CPU usage of the fog devices is higher in PTC in Amazon AWS experiments. It is in the range of 42%–94% (Fig. 12a). The distribution for baselines converges within 68% for first-fit and Foglets, within 67% for best-fit and within 70% for the mobility-based algorithm, respectively. The distribution of bandwidth usage of the fog devices is marginally higher in PTC than other baselines when experimented with over Amazon EC2. It is in the range of 53 KB/s–204 KB/s (Fig. 12b) for PTC. On the contrary, such distributions converge within 81 KB/s for the best-fit mechanism, within 121 KB/s for Foglets, within 77 KB/s for first-fit and within 78 KB/s for the mobility-based algorithm. The overhead in terms of the fog controller's

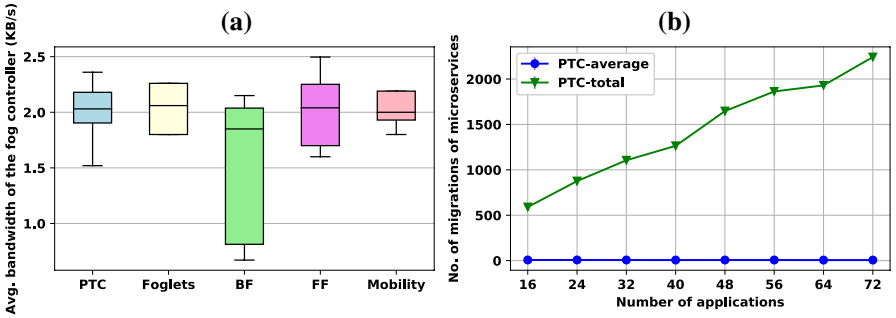


Fig. 14 **a** Distribution of average network bandwidth usage of the fog controller device and **b** Number of migrations of the micro-services

memory usage is shown in Fig. 13a. We have observed that the controller's average memory usage distribution is more in PTC in the Amazon EC2 experiments. It is in the range of 237 MB–798 MB (Fig. 13(a)). On the contrary, the fog controller's average memory usage distribution converges within 650 MB for first-fit, within 601 MB for Foglets, within 655 MB for best-fit and within 649 MB for the mobility-based algorithm.

Similar to our previous observations, the CPU usage of the controller is higher in PTC than other baselines and it is in the range of 65–97% (Fig. 13(b)). The distribution for baselines converges within 81% for mobility-based algorithm and 80% Foglets, respectively. It converges within 80% for best-fit and first-fit algorithm. The distribution of the average bandwidth distribution of the controller for PTC is in the range of 1.52 KB/s–2.36 KB/s (Fig. 14(a)). In contrary, the distribution for baselines converges within 2.15 KB/s for best-fit, within 2.49 KB/s for first-fit, within 2.26 KB/s for Foglets and within 2.19 KB/s for the mobility-based mechanism.

The number of migrations of the micro-services for the Amazon EC2 experiments is shown in Fig. 14(b). In these experiments, we observe that the total number of migrations increases with the number of applications. Also, the average number of micro-service migrations is in the range of 6.03 to 7.37.

7 Conclusion

With the widespread deployment of IoT-based services, the concept of fog computing can provide a useful solution for low-latency privacy-ensured data processing by utilizing the in-network processing capability of various devices. Although many existing works have focused on scheduling and deployment of application micro-services over fog nodes, the dynamicity of the fog devices' primary workloads is still a concern. In this paper, we have proposed PTC, a reinforcement learning-induced framework for continually monitoring the fog devices' primary workloads and accordingly dynamically deciding the execution platform for the application micro-services. PTC solves a hard time-varying optimization problem with Bayesian optimization, which dynamically finds a solution based on prior observations.

We have implemented PTC over a small-scale testbed setup and evaluated its performance over a large-scale emulation with Amazon EC2 clouds. The experiments confirm that PTC can reduce the application response time with better utilization of the fog device's excess resources, with the cost of a moderately complicated mechanism, which needs an additional edge server for execution.

We believe that the proposed framework can introduce dynamic execution of IoT data processing workloads by utilizing the fog devices' in-network computing facilities. Therefore, such a framework can be useful in various application scenarios, like smart home, smart manufacturing environments, etc. As future work, we plan to deploy the proposed framework with a complete IoT data collection and processing framework to build up an entire system by utilizing its performance. In future, we would like to perform evaluation of the proposed PTC framework with different IoT applications having different data processing needs.

References

1. Ahmad M, Amin MB, Hussain S, Kang BH, Cheong T, Lee S (2016) Health fog: a novel framework for health and wellness applications. J Supercomput 72(10):3677–3695
2. Ahmed A, Pierre G (2018) Docker container deployment in fog computing infrastructures. In: 2018 IEEE International Conference on Edge Computing (EDGE), pp. 1–8. IEEE
3. Alipourfard O, Liu HH, Chen J, Venkataraman S, Yu M, Zhang M (2017) Cherrypick: Adaptively unearthing the best cloud configurations for big data analytics. In: 14th USENIX Symposium on Networked Systems Design and Implementation (NSDI 17), pp. 469–482
4. Alturki B, Reiff-Marganiec S, Perera C, De S (2019) Exploring the effectiveness of service decomposition in fog computing architecture for the internet of things. IEEE Transactions on Sustainable Computing
5. Bernstein D (2014) Containers and cloud: from lxc to docker to kubernetes. IEEE Cloud Comput 1(3):81–84
6. Drezner Z, Wesolowsky GO (1983) Minimax and maximin facility location problems on a sphere. Naval Res Logistics Quart 30(2):305–312
7. Elgamal T, Sandur A, Nguyen P, Nahrstedt K, Agha G (2018) Droplet: distributed operator placement for iot applications spanning edge and cloud resources. In: 2018 IEEE 11th International Conference on Cloud Computing (CLOUD), pp. 1–8. IEEE
8. Gardner JR, Kusner MJ, Xu ZE, Weinberger KQ, Cunningham JP (2014) Bayesian optimization with inequality constraints. ICML 2014:937–945
9. Goethals T, De Turck F, Volckaert B (2020) Near real-time optimization of fog service placement for responsive edge computing. J Cloud Comput 9(1):1–17
10. Gonçalves D, Velasquez K, Curado M, Bittencourt L, Madeira E (2018) Proactive virtual machine migration in fog environments. In: 2018 IEEE Symposium on Computers and Communications (ISCC), pp. 00742–00745. IEEE
11. Gu L, Zeng D, Guo S, Barnawi A, Xiang Y (2015) Cost efficient resource management in fog computing supported medical cyber-physical system. IEEE Trans Emerg Top Comput 5(1):108–119
12. Javadzadeh G, Rahmani AM (2020) Fog computing applications in smart cities: a systematic survey. Wireless Netw 26(2):1433–1457
13. Kayal P, Liebeherr J (2019) Distributed service placement in fog computing: an iterative combinatorial auction approach. In: 2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS), pp. 2145–2156. IEEE
14. Kecskemeti G, Marosi AC, Kertesz A (2016) The entice approach to decompose monolithic services into microservices. In: 2016 International Conference on High Performance Computing & Simulation (HPCS), pp. 591–596. IEEE
15. Li DC, Huang CT, Tseng CW, Chou LD (2021) Fuzzy-based microservice resource management platform for edge computing in the internet of things. Sensors 21(11):3800

16. Li X, Wan J, Dai HN, Imran M, Xia M, Celesti A (2019) A hybrid computing solution and resource scheduling strategy for edge computing in smart manufacturing. IEEE Trans Industr Inf 15(7):4225–4234
17. Liao S, Wu J, Mumtaz S, Li J, Morello R, Guizani M (2020) Cognitive balance for fog computing resource in internet of things: an edge learning approach. IEEE Trans Mobile Comput
18. Mukherjee M, Shu L, Wang D (2018) Survey of fog computing: fundamental, network applications, and research challenges. IEEE Commun Surv Tutorials 20(3):1826–1857
19. Mutlag AA, Abd Ghani MK, Arunkumar Na, Mohammed MA, Mohd O (2019) Enabling technologies for fog computing in healthcare iot systems. Future Gener Comput Syst **90**, 62–78
20. Nadgowda S, Suneja S, Bila N, Isci C (2017) Voyager: Complete container state migration. In: 2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS), pp. 2137–2142. IEEE
21. Nath SB, Chattopadhyay S, Karmakar R, Addya SK, Chakraborty S, Ghosh SK (2019) Ptc: Pick-test-choose to place containerized micro-services in iot. In: 2019 IEEE Global Communications Conference (GLOBECOM), pp. 1–6. IEEE
22. Rossi F, Cardellini V, Presti FL (2019) Elastic deployment of software containers in geo-distributed computing environments. In: 2019 IEEE Symposium on Computers and Communications (ISCC), pp. 1–7. IEEE
23. Rossi F, Cardellini V, Presti FL, Nardelli M (2020) Geo-distributed efficient deployment of containers with kubernetes. Comput Commun 159:161–174
24. Saurez E, Hong K, Lillethun D, Ramachandran U, Ottenwälder B (2016)Incremental deployment and migration of geo-distributed situation awareness applications in the fog. In: Proceedings of the 10th ACM International Conference on Distributed and Event-based Systems, pp. 258–269
25. Singh SP, Nayyar A, Kumar R, Sharma A (2019) Fog computing: from architecture to edge computing and big data processing. J Supercomput 75(4):2070–2105
26. Snoek J, Larochelle H, Adams RP (2012) Practical bayesian optimization of machine learning algorithms. Adv Neural Inf Proc Syst 25:1
27. Souza VB, Masip-Bruin X, Marín-Tordera E, Sànchez-López S, Garcia J, Ren GJ, Jukan A, Ferrer AJ (2018) Towards a proper service placement in combined fog-to-cloud (f2c) architectures. Futur Gener Comput Syst 87:1–15
28. Stévant B, Pazat JL, Blanc A (2018) Optimizing the performance of a microservice-based application deployed on user-provided devices. In: 2018 17th International Symposium on Parallel and Distributed Computing (ISPDC), pp. 133–140. IEEE
29. Taherizadeh S, Apostolou D, Verginadis Y, Grobelnik M, Mentzas G (2021) A semantic model for interchangeable microservices in cloud continuum computing. Information 12(1):40
30. Taherizadeh S, Stankovski V, Grobelnik M (2018) A capillary computing architecture for dynamic internet of things: Orchestration of microservices from edge devices to fog and cloud providers. Sensors 18(9):2938
31. Taneja M, Davy A (2017) Resource aware placement of iot application modules in fog-cloud computing paradigm. In: 2017 IFIP/IEEE Symposium on Integrated Network and Service Management (IM), pp. 1222–1228. IEEE
32. Wang S, Guo Y, Zhang N, Yang P, Zhou A, Shen XS (2019) Delay-aware microservice coordination in mobile edge computing: a reinforcement learning approach. IEEE Trans Mobile Comput
33. Wang W, Zhao Y, Tornatore M, Gupta A, Zhang J, Mukherjee B (2017) Virtual machine placement and workload assignment for mobile edge computing. In: 2017 IEEE 6th International Conference on Cloud Networking (CloudNet), pp. 1–6. IEEE
34. Yigitoglu E, Mohamed M, Liu L, Ludwig H (2017) Foggy: a framework for continuous automated iot application deployment in fog computing. In: 2017 IEEE International Conference on AI & Mobile Services (AIMS), pp. 38–45. IEEE
35. Yin L, Luo J, Luo H (2018) Tasks scheduling and resource allocation in fog computing based on containers for smart manufacturing. IEEE Trans Industr Inf 14(10):4712–4721

Authors and Affiliations

Shubha Brata Nath[1] · Subhrendu Chattopadhyay[2] · Raja Karmakar[3] · Sourav Kanti Addya[4] · Sandip Chakraborty[1] · Soumya K Ghosh[1]

Shubha Brata Nath
nath.shubha@gmail.com

Subhrendu Chattopadhyay
subhrendu@iitg.ac.in

Raja Karmakar
rkarmakar.tict@gmail.com

Sourav Kanti Addya
kanti.sourav@gmail.com

Soumya K Ghosh
skg@cse.iitkgp.ac.in

1 Department of Computer Science and Engineering, Indian Institute of Technology Kharagpur, 721302 Kharagpur, India

2 Department of Computer Science and Engineering, Indian Institute of Technology Guwahati, 781039 Guwahati, India

3 Techno International New Town, New Town 700156, India

4 Department of Computer Science and Engineering, National Institute of Technology Karnataka, Surathkal 575025, India

DisProTrack: Distributed Provenance Tracking over Serverless Applications

Utkalika Satapathy*, Rishabh Thakur*, Subhrendu Chattopadhyay†, Sandip Chakraborty*
*IIT Kharagpur, Kharagpur, India 721302 †IDRBT, Hyderabad, India 500057

***Abstract*—Provenance tracking has been widely used in the recent literature to debug system vulnerabilities and find the root causes behind faults, errors, or crashes over a running system. However, the existing approaches primarily developed graph-based models for provenance tracking over monolithic applications running directly over the operating system kernel. In contrast, the modern DevOps-based service-oriented architecture relies on distributed platforms, like serverless computing that uses container-based sandboxing over the kernel. Provenance tracking over such a distributed micro-service architecture is challenging, as the application and system logs are generated asynchronously and follow heterogeneous nomenclature and logging formats. This paper develops a novel approach to combining system and micro-services logs together to generate a Universal Provenance Graph (UPG) that can be used for provenance tracking over serverless architecture. We develop a Loadable Kernel Module (LKM) for runtime unit identification over the logs by intercepting the system calls with the help from the control flow graphs over the static application binaries. Finally, we design a regular expression-based log optimization method for reverse query parsing over the generated UPG. A thorough evaluation of the proposed UPG model with different benchmarked serverless applications shows the system's effectiveness.**

***Index Terms*—Distributed provenance tracking,**

I. Introduction

Modern service-oriented architecture adopts DevOps [1], [2] practices and technologies to provide Software as a service (SaaS) [3] by leveraging distributed cloud infrastructure. Service deployment on top of the cloud widely adopts serverless computing (SLC) [4] to reduce operational expenditure whenever the service computations are stateless, elastic, and possibly distributed. Micro-services deployed on top of SLC architecture provide an abstraction of the underlying infrastructure where the developer can write, deploy and execute the code without configuring and managing the shared environment [4]–[7]. However, the available serverless-specific industry solutions [8]–[10] provide limited support for error reporting, execution tracing, and provenance tracking. Consequently, developers can only provide little attention to log vital forensic information. Some of the third-party observability tools [11]–[14] support distributed tracing as well as cost analysis features by instrumenting the source codes. However, these tools only support applications developed using a particular programming language. So, it is difficult to analyze the actual behavior of these micro-services.

Provenance Graphs: The non-invasive [1] frameworks rely on the logs generated by applications to identify the execution states. Since most of the production-grade micro-services are chosen from an available stable release, the executable files already contain meaningful log messages that can be used to identify event handling loops. In the domain of system security, provenance data is the metadata of a process that records the details of the origin and the history of modification or transformation that happened over time throughout its lifecycle [15]–[20]. The graph generated from this information is called the *provenance graph* of a process which is a causal graph that stores the dependencies between system subjects (e.g., processes) and system objects (e.g., files, network sockets). For example, an application event may generate a separate application log, error log, and operating system calls specific log, etc. In this context, a provenance graph is a directed acyclic graph (DAG) where individual log entries are the nodes, and the edges represent the causality relationship between the log entries. During attack investigation, an administrator queries this graph to find out the root cause and ramifications of an event. While a malicious entity performs some illegal events, the corresponding system logs are recorded as the provenance data. For example, when a compromised process tries to open a sensitive file, the OS level provenance can record the file-open activity, which can be referred for vulnerability analysis whenever an attack is detected in the system [21], [22]. Real-world enterprises widely use kernel or OS level information (logs) to perform provenance analysis to monitor their systems and identify the malicious events performed back in time.

A. Limitations of Existing Works and the Research Challenges

There have been several works that attempted to generate the provenance graphs by combining system and application logs together [16]–[20], [23]. However, these works primarily considered a monolithic application running directly over the Kernel, and thus combining the system log with the application log is not difficult. In contrast, the individual log files for each micro-service over an SLC application are physically distributed across the entire eco-system, which makes the generation of the provenance graph non-trivial. In such a scenario, a Universal Provenance Graph (UPG) that combines the interactions among all the micro-services can provide a meaningful platform for distributed provenance tracking. A few existing works [20], [24] have tried to address the issue of encoding all forensically-relevant causal dependencies regardless of their origin. Nevertheless, we have some non-trivial

[1]The non-invasive tools neither inject any piece of code inside the micro-service/container instances nor injects any special services into the SLC platform.

challenges in constructing a UPG for an SLC application.

• Challenge (1) – *Combining application logs from different micro-services:* Different micro-services generate separate logs that vary across the format for log messages, naming of the events and process descriptors, timestamp formatting, etc. Combining these logs towards generating the UPG is non-trivial as they may result in confounding nodes and edges in the graph.

• Challenge (2) – *Combining the system log with the application logs:* The next challenge comes in terms of combining the application logs with the system log (kernel `audit log`). The first issue is that the container-based sandboxing uses a common process identifier (`pid`) space; thus, mapping the kernel process logs with the micro-services event logs is not straight-forward, particularly when a micro-service run a multi-threaded process.

• Challenge (3) – *Identification of execution units:* As different micro-services may come from different production endpoints, there exists heterogeneity in terms of their implementation standards. Thus it is non-trivial to identify the functions or execution units that generate a specific entry in the log. In the same context, it is also difficult to identify the event handling loops, as the loops might produce asynchronous log entries. This problem magnifies in the case of SLC as the micro-services are deployed in different sandboxed environments.

• Challenge (4) – *Dependency explosion and handling confounding root causes:* To find out the root cause behind an event, the system administrator needs to execute a reverse query on the UPG. The results obtained from the reverse query can be multiple due to partial matching of the input query string. This results in more than one root cause behind a query event. Further, due to plausible circular dependencies among the micro-services, the resultant UPG might not be a DAG. Therefore, it is non-trivial to track all the causal paths behind an event.

Owing to the above challenges, in this paper, we develop a provenance tracing system, `DisProTrack` that generates a UPG which helps in root cause analysis of an event running on serverless by combining the system provenance with application's container logs. The core idea behind `DisProTrack` is to judicially use the applications' control flow graph (CFG) to avoid the runtime log tracking complexities.

B. Our Contributions

In contrast to the existing works, our contributions in this paper are as follows.

1. Design of the UPG from application and system logs: We implement a static analyzer module that generates the application-specific *Log Message String - Control Flow Graph* (LMS-CFG) from the application binaries. The LMS-CFG provides a profile of the application. We provide a novel approach for constructing a UPG from application logs and system logs using the LMS-CFG profiles for different application micro-services.

2. Runtime execution unit identification: We develop a Linux LKM (loadable kernel module) which can intercept the system calls generated during execution time to identify the semantic relationship between the system logs and the application logs. Furthermore, we propose a heuristic to segregate execution units which are challenging in a distributed system. Our proposed heuristic identifies the system calls (syscalls) to mark the application's event handling loops by tracing back the application binaries. These event handling loops can be refereed during the runtime partitioning of execution units across the micro-services.

3. Utilization of Regular Expression to improve search efficacy: Instead of storing the raw log messages in the UPG, we propose conversion and storage of an equivalent regular expression. This method improves the matching accuracy of log messages during the investigation phase and reduces the runtime search complexity by providing a faster response time. This method also reduces dependency explosion by decreasing the number of nodes in the generated UPG.

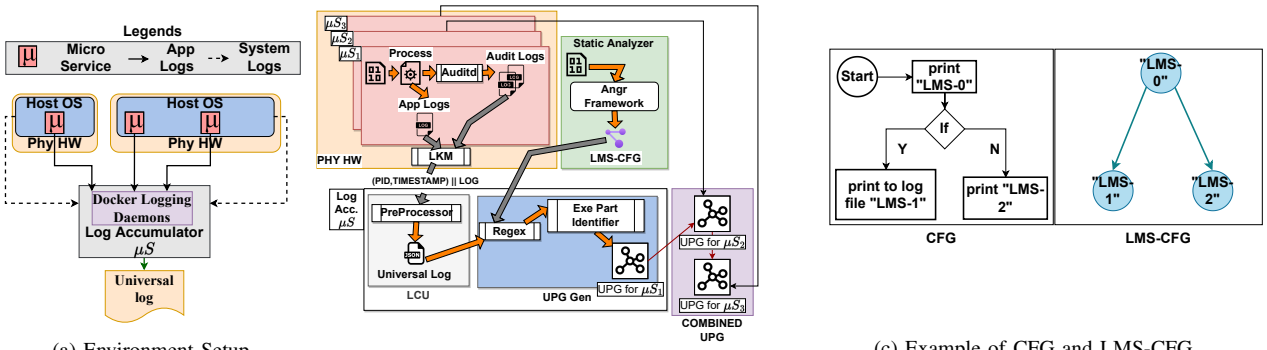
4. Implementation and evaluation: We have implemented the proposed framework, which can be deployed as a micro-service on top of the SLC without instrumenting the source code of the applications. We have made the implementation open-sourced[2]. Based on the experimental evaluation of `DisProTrack` with several benchmarked SLC applications, we found that the proposed method works well for identifying adversary activities. The framework has a minimal memory footprint (in the order of KB) and responds within $20s$-$30s$. The efficiency and efficacy of `DisProTrack` have also been tested with a proof-of-concept SLC application scenario.

II. Related Work

Existing third party log collection tools like, `FUSE` [25], `LSM` [26], `CamFlow` [27], `SPADE` [28], etc., allow hooking the kernel level objects and system calls; however, these methods require instrumentation of the OS. Additionally, the collection of system-level provenance in a containerized or serverless environment is difficult due to the micro-services' distributed and minimalistic deployed nature. Therefore, the existing threat detection and investigations with system-level provenance graph using kernel audit-log data, such as, `ETW` [29], `SLEUTH` [30], `Pagoda` [31], `POIROT` [32], `HOLMES` [33], `WATSON` [34], etc., do not suit an SLC environment. One common challenge for causality analysis with system provenance graph is the *"Dependency Explosion"* problem [35], [36] where too many *"root causes"* are behind one suspicious event. This dependency explosion increases the probability of *"security alert fatigue"* and *"missing threats"*.

`BEEP` [36] and `OMEGALOG` [16] have addressed dependency explosion problem by increasing input and output edge identification accuracy in the provenance graph. Other methods

[2]https://anonymous.4open.science/r/Project_ALV_2022-CEFD/ (Accessed: March 23, 2024)



(a) Environment Setup

(b) DisProTrack Execution Flow

(c) Example of CFG and LMS-CFG

Fig. 1: DisProTrack Overview

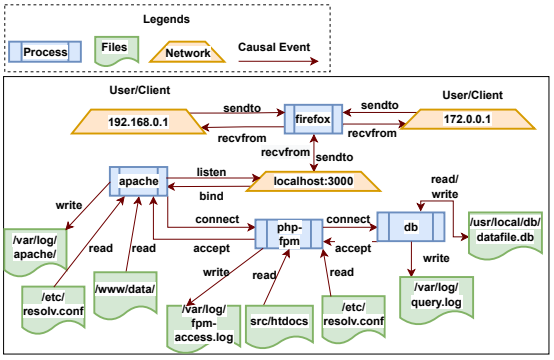


Fig. 2: An Example of System Provenance

like ETW [29], LogGC [37], MPI [35], ProTracer [38], etc., used execution partitioning to reduce the dependency explosion. However, most of these works require some instrumentation over the application source code. On the other hand, ALCHEMIST [20] and OMEGALOG used a combination of application-specific logs with system-level logs to track the information flow more accurately. In the same line, UIScope [39] proposed an instrumentation free, execution partition-based causality analysis for attack investigation system. However, the existing works are targeted toward monolithic systems and can not be deployed to secure SLC applications. Although, a few existing tools like, X-trace [40] and PivotTracing [41] are used to instrument SLC applications, they do not provide non-invasive property. Similarly, [11]–[14] are reliant on the used programming languages of the micro-service applications; thus lack flexibility [24]. The absence of language-independent, non-invasive causality analysis frameworks for SLC has motivated us to design DisProTrack.

III. DisProTrack Overview

The proposed DisProTrack is an open source provenance tracking system for containerized SLC as shown in Fig. 1a. DisProTrack accumulates the system and application logs of all the micro-service containers (μS) to generate the underlying directed edge-labeled provenance graph. The nodes of the provenance graph represent the system artifacts, for example, processes, socket connections, files, etc. The edges represent the causal dependencies between the nodes. Each edge is labeled with the generated system call (syscall) events (e.g. *read*, *write*, *connect*, *exec*, etc.) as shown in Fig. 2. The accumulated logs are merged together to create a *"universal log"* which is further used to identify the nodes of the provenance graph. Additionally, the framework also analyzes the CFG of the individual applications to understand the event handling loops before the deployment. So, the provenance graph generation requires two phases and, depending on that, DisProTrack is sub-divided into two major components; (a) *static analyzer* and (b) *runtime engine* as shown in Fig. 1b.

A. DisProTrack Static Analyzer

The static analyzer module analyzes the application executables and generates a semantic relationship between multiple *Log Message String* (LMS). Here, we define a LMS as a string present in the executable responsible for printing some log message. Typical LMSes contains format specifiers, error codes, debug level identifiers, etc. Since we only require the causal relationships between LMSes, the static analyzer identifies only the *Log Message Generating Functions* (LMGF).

Definition 1 (Log Message Generating Functions)**.** We define a LMGF as a library function that is directly used for printing a LMSes in either terminals, specific log store files, or log databases.

For example, consider the following C code snippet with a popular logging library Log4C.

```
log4c_category_log(NULL,
LOG4C_PRIORITY_ERROR,"Hello World!");
```

Here the LMS is Hello World! and the LMGF is log4c_category_log. More details about LMGF is described in Section IV-A.

Definition 2 (Log Message String CFG)**.** A LMS-CFG is formally defined as a directed graph $G' = (V', E')$ where $\forall_{e'_{i,j} \in E'} e'_{i,j} = (v'_i, v'_j) : v'_i, v'_j \in V'$ represents a directed edge between v'_i, v'_j where each $v' \in V'$ represents an LMGF. G' is constructed from CFG $G = (V, E)$ such that, $V' \subseteq V$ and,

$\underset{e' \in E'}{\exists} e' = (v'_i, v'_k) : v'_i, v'_j, v'_k \in V', \nexists_{v'_j} v'_j \in \mathfrak{P}(G, v'_i, v'_k)$. In this case $\mathfrak{P}(G, v'_i, v'_k)$ represents the directed path from v'_i to v'_k in G.

An example of a CFG and the corresponding LMS-CFG is provided in Fig. 1c where the constructed LMS-CFG contains only the LMS nodes from the CFG.

1) Contextualization of application log and system log (handling Challenges 1 & 2): The LMS-CFG is stored separately and frequently consulted by the runtime engine. During the execution, when the different levels of logs are generated, the constructed LMS-CFG is used to understand the relationship among those log messages. Based on the relationship, the logs coming from different sources are combined together (details in Section IV-B2).

B. `DisProTrack` Execution Path (Runtime Analyzer)

During the execution of the system, the applications generate multiple logs depending on the user's activities. So the task of the runtime engine is to collect and contextualize the log messages generated at the systems and application levels. We assume that the micro-services containers have `auditd` [42] installed for monitoring system-level logs. This assumption does not violate the *"without instrumentation of the application source code"* constraint, as it is straightforward to add an `auditd` layer to the existing containers without knowing anything about the application source codes.

1) Tracing the execution units from processes belonging to different micro-services (handling Challenge 3): To avoid confusion during the contextualization process of the accumulated logs, we append the Process ID (PID) of the process responsible for generating the log and the timestamp to add the semantic context to individual log entries. For this purpose, we develop a Loadable Kernel Module (LKM) which intercepts all the write syscalls caused by log printing functions to extract the PID information and timestamp of the system and append it to the log preamble. This LKM is deployed in the bare metal infrastructure where the containers are executing (details in Section IV-B2).

2) Dependency explosion and handling confounding root causes (handling Challenge 4): During the execution of the applications, the runtime engine periodically fetches the marked log entries from the micro-services. It generates the UPG after consulting the LMS-CFG. The LMS-CFG provides a relationship between the applications and file, which is exploited to construct UPG as depicted in Fig. 2. The details of the UPG construction procedure are described in the next section. The generated UPG is consulted by the system administrators for the system provenance tracking. The root cause of any suspicious log entry can be identified by backtracing the UPG (details in Section IV-C).

IV. Components of DisProTrack

In this section, we describe the design of individual components of `DisProTrack`. As mentioned previously, `DisProTrack` is subdivided into two parts; (a) *Static Analyzer*, and (b) *Runtime Engine*.

A. Static Analyzer

The static analyzer takes individual micro-service's application binaries as input and constructs the corresponding LMS-CFG for that micro-service application. A typical application will contain a set of statements to print the LMSes with syscalls in between. In our proposed framework, we load the executable application binary and use the python `Angr` module [43], [44] to identify the CFG from it. The generated CFG is a directed graph having the basic instruction blocks (syscalls, printing of LMSes, etc.) as nodes, and the edges of the graph represent jump/call/return statements from one block to another. Let the CFG be represented as $G = (V, E)$, where V and E are the nodes and the edges of the CFG, respectively. We then use the same `Angr` module to extract all the nodes $\mathbb{F}$ from the CFG $G(V, E)$ corresponding to various function calls. One significant issue with the generated CFG is that it does not always provide the complete graph due to the missing hardware-dependent features and system call information. Therefore, in this case, we only concentrate on the LMGF. Typically the stable versions of the applications use standard LMGF (e.g. `printf`, `log4c`, `syslog`, etc.). Let $\mathcal{L}$ be a list of standard LMGF names. We find $\mathbb{L} \subseteq \mathbb{F}$ where $\mathbb{L}$ contains the LMGFs from $\mathcal{L}$. We construct a LMS-CFG $G'(V', E')$ from $G(V, E)$ with the help of $\mathbb{L}$.

We propose Algorithm 1 to convert $G = (V, E)$ to $G' = (V', E')$ for a given $\mathbb{L}$ as the input. Each node in G represents a sequence of instructions. If a node v contains a LMGF name, then the algorithm extracts the arguments of the LMGF, which has been defined as a LMS apriori. For example, for a given LMGF `fprintf`, consider the following log entry function.

"`fprintf(stderr, "AH00526: Syntax error on line %d of %s:");`"

In this case, the algorithm extracts the LMS as `"AH00526: Syntax error on line %d of %s:"`, which is the constant string reference passed as an argument to the LMGF.

During the construction of LMS-CFG, we need to identify the caller functions of the LMGF, which is a time-consuming operation. Therefore, we limit the depth of backward tracing up to a certain threshold (BT), as shown in Algorithm 1:Line 13. The optimal value of BT depends on the complexity of the generated CFG, which we shall discuss during the experimental evaluation (Section V-B1). For accurate identification of the LMSes, we need to avoid the programming language-specific format specifiers. For that, we replace the format specifier of LMS with equivalent regular expressions (see Algorithm 1:Line 10). For example, consider the LMS as shown earlier: "`AH00526: Syntax error on line %d of %s:`". The regular expression equivalent to this LMS becomes "`AH00526: Syntax error on line -?[0-9]+ of .*:`", where `%d` and `%s` are replaced with `[0-9]+` and `.*`, respectively. Additionally, we mark the starting and ending LMS positions of the event handling loops with a flag. This generated and marked LMS-CFG is used by *Runtime Engine* explained next.

Algorithm 1: CFG to LMS-CFG Generation

Procedure `Main`
  Input: $G(V, E)$: CFG, $\mathbb{L}$: Set of LMGFs in $G(V, E)$
  Output: LMS-CFG $G' = (V', E')$
  Initialization:
    $V' \leftarrow \emptyset, E' \leftarrow \emptyset$
  for $v \in \mathbb{L}$ **do**
    /* The Angr tools return whether a node is a loop */
    if *v has a loop* **then**
      /* For a loop, we need to find out all the LMSes ($\mathbb{A}$) that are printed through the loop. */
      $\mathbb{A} \leftarrow$ `BackTraceOptimization`$(BT_{th}, G(V, E), v)$;
      /* Construct the subgraph $G_A(V_A, E_A)$ for A */
      $G_A(V_A, E_A) \leftarrow$ `CreateSubGraph`(A_i);
      $V' \leftarrow V' \cup V_A$;
      $E' \leftarrow E' \cup E_A$;
  for $v' \in V'$ **do**
    Identify format specifiers in v' and replace them with suitable Regular Expressions;
  return $G'(V', E')$;

Function `BackTraceOptimization`
  Input: BT: Backtrace Threshold, $G(V, E)$: CFG, v: A node from $\mathbb{L}$ having a loop
  Output: $\mathbb{A}$: A set of LMSes in the loop
  Initialization:
    $\mathbb{A} \leftarrow \emptyset$
  if *v has a set of LMSes $\{l_0, \cdots l_k\}$ printed through the loop with syscalls in between the LMSes* **then**
    /* We consider the loops having a syscall and associated LMSes */
    $\mathbb{A} \leftarrow \mathbb{A} \cup \{l_0, \cdots, l_k\}$;
    return $\mathbb{A}$;
  else
    /* The loop within v does not print any LMSes */
    do
      /* Backtrack to the LMGF that called v, until the backtrack threshold BT is reached. */
      if *$\langle \bar{v} \rightarrow v \rangle$ is a directed edge in $G(V, E)$* **then**
        if *$\bar{v}$ has a loop with a syscall and a set of LMSes $\{l_0, \cdots l_k\}$* **then**
          $\mathbb{A} \leftarrow \mathbb{A} \cup \{l_0, \cdots, l_k\}$;
          return $\mathbb{A}$;
      $v \leftarrow \bar{v}$;
      $BT \leftarrow BT - 1$;
    while $BT > 0$;
  return $\mathbb{A}$;

Function `CreateSubGraph`
  Input: $\mathbb{A}$
  Output: $G_A(V_A, E_A)$: A Subgraph generated from $\mathbb{A}$
  Initialization:
    $V_A \leftarrow \emptyset, E_A \leftarrow \emptyset$
  foreach $\{l_i, l_{i+1}\} \in \mathbb{A}$ **do**
    $V_A \leftarrow V_A \cup \{l_i, l_{i+1}\}$;
    $E_A \leftarrow E_A \cup \{\langle l_i \rightarrow l_{i+1} \rangle\}$;
  return $G_A(V_A, E_A)$;

B. Runtime Engine

We design `DisProTrack` Runtime Engine as a micro-service that can be deployed in the serverless platform. Since most of the serverless functions are deployed using multiple containers, log collection and analysis are challenging. `DisProTrack` is concerned about two types of logs; (i) Logs generated by the applications (i.e., inside the container) and (ii) System and/or serverless daemon logs (i.e., logs from the physical servers systems) – we call them Syslogs. The major challenge of obtaining a container's internal logs is that as the process spaces of the containers and the host system are isolated, identification of processes and syscalls become difficult. To avoid such issues, we use an audit daemon in each container by deploying a separate image layer[3] over the containers. The audit daemon is used to track the syscalls generated by the applications inside the containers.

However, audit logs provide the container internal PID, which might conflict with the audit logs obtained from different containers. The conflict must be resolved before the aggregation of the system log and application log. Therefore, we develop a LKM which intercepts LMS before it is printed in the log file and appends a unique tag (which is a combination of container ID, PID, and timestamp) to each LMS entry. This unique tag is used to establish a relationship by adding semantic context among the LMS.

The scope of operation for the runtime engine starts whenever the applications start generating logs. We have segregated Runtime Engine into three submodules; (a) Log accumulator, (b) Log processor, and (c) Provenance builder, which are described as follows.

1) Log Accumulator: Once the log files are generated, the *Log Accumulator* module periodically pulls the log files from all levels and performs operations on them to correlate them with the events. The non-persistent and ephemeral nature of micro-services implies the risk of data loss or the loss of logs generated during the execution phase of the container lifecycle when the container shuts down. Therefore, the proposed module periodically pulls the logs. To prevent data loss due to "`SIGKILL`", we deploy a signal handler inside the deployed image layer to instruct the container to save the logs in a persistent data volume.

2) Log Processor: Once the logs are accumulated, the *Log Processor* module aggregates the logs collected from various sources. However, simple concatenation of log files does not preserve the semantics relationship. Therefore, we convert the text-based log files into an equivalent `JSON` format. From the formatted application log files, the PID of the application is extracted from the tag generated by the LKM. Using the PID, the syscalls are identified from the audit logs. Now the LMS and the syscall-generated logs are merged to create an Application-Specific Common Log (ASCL) file such that the application logs appear just before the corresponding Syslogs.

C. Provenance Builder

At the runtime, the *Provenance Builder* takes the ASCL file and LMS-CFG of a process as input and constructs the corresponding UPG component for that process. The ASCL file contains interleaved application-level logs ($\langle pid, ts, lms \rangle$) and Syslog ($\langle pid, ts, syscall, path, exe \rangle$) entries. Using Algorithm 2, we identify the execution units in the ASCL file with the help of LMS-CFG, where an execution unit represents a

[3]https://docs.docker.com/storage/storagedriver/ (Accessed: March 23, 2024)

Algorithm 2: UPG Construction

1 **Procedure** `Main`
  Input: $G' = (V', E')$: LMS-CFG, $\mathbb{F}_{pid}$: ASCL file for process pid
  Output: $G^U_{pid} = (V^U_{pid}, E^U_{pid})$: UPG component for process pid
2 **Initialization:**
3 $V^U_{pid} \leftarrow \emptyset$, $E^U_{pid} \leftarrow \emptyset$, $\mathbb{U}_{pid} \leftarrow \emptyset$, $\mathbb{E}_{pid} \leftarrow \emptyset$, $\mathbb{G}_{pid} \leftarrow \emptyset$;
  /* $\mathbb{U}_{pid}$: An execution unit denoting the set of LMSes matched with V' */
  /* $\mathbb{E}_{pid}$: Set of system logs corresponding to an execution unit */
  /* $\mathbb{G}_{pid}$: Set of all $\mathbb{E}_{pid}$ from $\mathbb{F}_{pid}$ */
4 $end\_unit \leftarrow$ false /* tracks execution units */
5 **foreach** e *in* $\mathbb{F}_{pid}$ **do**
  /* e can be an application-level entry $\langle pid, ts, lms\rangle$ or a syslog entry $\langle pid, ts, syscall, path, exe\rangle$ */
  /* pid: Process ID, ts: Log timestamp */
  /* lms: LMS, $syscall$: Syscall in log */
  /* $path$: System object (dir, socket, etc.) accessed by the executable */
  /* exe: Name of the executable */
6 **if** e *is an application-level entry* **then**
7 **if** e *is the first entry in* $\mathbb{F}$ **then**
  /* Perform a regex match to find a node n from $G'(V', E')$ which matches with e. n denotes the current state of the search. */
8 $n \leftarrow$ `FindLMSinGraph`(G', e);
9 $\mathbb{U}_{pid} \leftarrow \mathbb{U}_{pid} \cup \{n\}$
10 **else**
  /* Perform a regex match to find a node in the neighbor of n over the graph $G'(V', E')$; the new node becomes the current state of the search */
11 $n \leftarrow$ `MatchNeighbourNodes`(G', n, e);
12 $\mathbb{U}_{pid} \leftarrow \mathbb{U}_{pid} \cup n$;
13 **if** n *is a leaf node in* $G'(V', E')$ **then**
14 $end\_unit \leftarrow$ true /* Seen a block of LMSes logged by the application from a LMGF */
15 **if** end_unit *is true* **then**
16 $\mathbb{E}_{pid} \leftarrow \mathbb{E}_{pid} \cup e$;
17 $end\_unit \leftarrow$ false /* tracked syslogs for an execution unit */
18 $\mathbb{G}_{pid} \leftarrow \mathbb{G}_{pid} \cup \mathbb{E}_{pid}$;
19 $\mathbb{E}_{pid} \leftarrow \emptyset$
20 **else**
  /* e is a syslog entry and end_unit is false */
21 $\mathbb{E}_{pid} \leftarrow \mathbb{E}_{pid} \cup e$;
  /* Tracked all syslogs for individual execution units, now construct the UPG */
22 $partition \leftarrow 0$ /* keep tracks of individual execution units */
23 **foreach** $\mathbb{E}_{pid} \in \mathbb{G}_{pid}$ **do**
24 $partition \leftarrow partition + 1$;
25 **foreach** $e \in \mathbb{E}_{pid}$ **do**
26 **if** e *has a valid* exe *and* $syscall$ **then**
27 $V^U_{pid} \leftarrow V^U_{pid} \cup \{e.exe\}$ /* the name of the executable application binary becomes a node */
28 $V^U_{pid} \leftarrow V^U_{pid} \cup \{partition||e.path\}$ /* the $partition$ value appended with the object path accessed in e becomes the second node */
29 $E^U_{pid} \leftarrow E^U_{pid} \cup \{\langle e.exe \rightarrow partition||e.path, e.syscall\rangle\}$ /* an edge is added between the above two nodes with $e.syscall$ as the edge label */

sequence of LMSes execution of a process. In this heuristic, we intelligently apply the LMS-CFG to mark the end of an execution unit by using the leaf nodes of the graph. In this heuristic, we assume that the micro-services are *weakly time-synchronized* with a small time drift λ. The bound on λ depends on how frequently the log messages from two different execution units are getting printed. In reality, λ can be in the order of a few hundred milliseconds as printing the logs too frequently is also an overhead for an application.

Extract Execution Units with the help from LMS-CFGs of application micro-services. Initially, the execution unit is empty. When the algorithm encounters an application-level log entry from the file, which also happens to be the first entry, it performs a regular expression matching on the LMS-CFG to find a node with a match of that entry. If a valid match, say, n, is found, then n becomes the current state. For the rest of the log entries, it performs the regular expression matching with the neighbors of n from the LMS-CFG to find the next state. This step is repeated for all the application-level entries till we find n as a leaf node in the LMS-CFG, which denotes an end unit of the execution unit. The intermediate Syslog entries are added to their respective PID's execution unit E_{pid}. When the end unit becomes true, it implies a block of LMSes has been printed along with a syscall execution. Hence, we save the state of this execution unit in a set G_{pid} and make the execution unit E_{pid} empty for the next set of LMSes to be added. We also set the end unit flag to false.

Interconnect execution units. Once all the syslogs for an execution unit is extracted, Algorithm 2 (Line: 23-29) constructs the UPG for that execution unit G_{pid}. We keep track of individual execution units in G_{pid} using a variable called $partition$. For every execution partition in G_{pid}, if an entry e contains exe and $syscall$, then the nodes of the UPG are – (i) the name of the executable application binary ($e.exe$), and (ii) the system objects such as files, socket, directory, etc. ($e.path$) appended with the $partition$ value. The edges are added between the above two nodes, where the corresponding Syscall denotes the edge labels from the Syslog entry ($e.syscall$). The resulting UPG is the union of all the UPG components constructed for each PID.

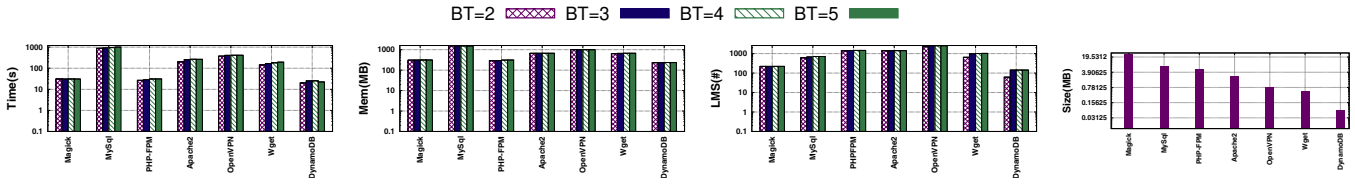
V. Performance Evaluation

The objective of our experimentation is two-fold as follows.

1) Since `DisProTrack` is targeted for serverless applications; resource overhead is a major concern. Therefore, experimentally we want to understand the resource overhead of the framework.
2) We also want to understand how effective `DisProTrack` is for identifying malicious activities.

For experimental analysis, we have implemented `DisProTrack`[4] and executed it in a testbed deployed in our lab. The implementation details are as follows.

[4]https://anonymous.4open.science/r/Project_ALV_2022-CEFD/



(a) Required Time (b) Memory Consumption (c) LMS Found (d) Application Size

Fig. 3: Static Analysis – The Y-axes are in logarithmic scale

A. Experimental Setup

The experiments are executed on a workstation having Intel(R) Xeon(R) Gold 5118 CPU @ 2.30GHz dual-core processor and 32 GB of memory. To ensure bare metal infrastructural abstraction, we use multiple Virtual Machine (VM) instances running with Ubuntu 22.04 LTS with Kernel version $5.15.0-39$-generic. The compute functions are deployed using Docker[5] across the VMs. We use docker-swarm[6] for container management where one container instance is deployed as the manager. For communication, we use an overlay network driver to ensure direct connectivity among the containers. During our experimentation, we have used the standard docker images of the applications collected from `Dockerhub`[7] with an added image layer of `auditd` as described in Section IV-B1. On the other hand, the proposed static analyzer and runtime engine are deployed as a containerized micro-service.

B. Resource Utilization

We conducted experiments to analyze the resource utilization and overhead imposed by `DisProTrack`, which has a two-fold view – (i) the overhead of the static analyzer, which is a one-time event for every deployment scenario, and (ii) the overhead of the runtime engine, which is a periodic event. Therefore, we present different sets of experiments to understand these overheads as follows. Each experiment is repeated 10 times, and the mean is shown in the plots.

1) Overhead of Static analyzer: To understand the resource utilization overhead during static analysis, we have considered 7 different applications as given in Table I.

TABLE I: List of Benchmarked Applications

Name	Type of Application
`Apache Webserver`	Together popular as LAMP-Stack and widely used for web service deployment
`PHP-FPM`	
`MySql`	
`DynamoDB`	A noSQL based database used in Amazon Lambda stream processing usecase
`ImageMagick`	An image processing framework can be used for various Amazon Lambda file processing usecases
`OpenVPN`	A popular Secure IP tunnel daemon
`Wget`	Linux network downloader

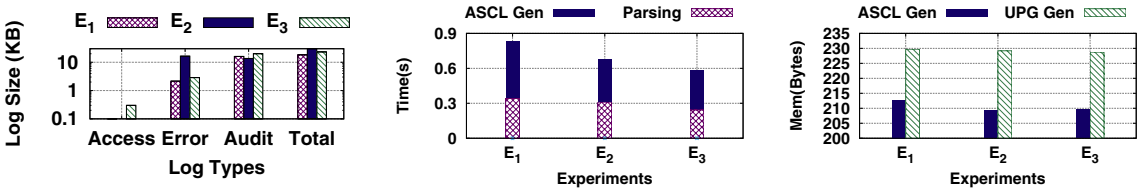
Based on the obtained results (Figs. 3a and 3b), we observe that the static analysis for `MySql` takes the longest time to complete and needs a greater amount of memory. The time and memory requirements increase when the BT increases. This is justified as the value of BT increases, the number of backtraces also increases (as mentioned in Section IV-A). However, an increase in the number of backtraces can identify more number of LMSes as shown in Fig. 3c. We also observe that even though the size of `Magick` is greater than the size of `MySql` (Fig. 3d), it takes more time and memory for `MySql` to trace the LMSes. This is due to the nature of the program control instructions used by developers while developing the application. Hence, it takes more time and memory to complete the backtrace in the presence of higher number of indirect branch instructions. For the same reason, `PHP-FPM` takes less amount of time and memory than `MySql`; however, it identifies more number of LMSes.

2) Overhead of Runtime Engine: Since the runtime engine works for a pipeline of micro-services, we execute multiple experiments on a web service application. Our deployed LAMP stack web service is composed of 3 micro-services (similar to Fig. 2); (a) [μ_1]:`Apache` based web server, (b) [μ_2]: `PHP-FHM` for server-side request handling, and (c) [μ_3]: `MySQL` for handling queries generated by the `PHP-FHM`. Each HTTP request incident on μ_1 forwards a `FastCGI` requests to μ_2. μ_2 executes the handler functions and accesses the database deployed in μ_3 for dynamic content. Once the page is constructed, μ_1 returns it as a response to the client. Specifically, we have hosted an application authorization portal where μ_1 interacts with the μ_2 via μ_3 to validate the user credentials. Upon receiving the valid credentials, μ_1 redirects from the login page to a user-specific home page. Otherwise, it remains on the same page with an error message pop-up. On top of the authorization service, we consider three experimental login scenarios as follows.

- E_1 New users register themselves, and the details are updated in the database. Once registration is successful, the user tries to log in and then log out of the application with valid credentials.
- E_2 A user logs in with a valid credential and goes to the home page. Once logged in, they try to reset the password and re-login with a new password.
- E_3 A user tries to log in to the deployed web service. On the homepage, they click on a link that triggers a background script and redirects the user to a different IP.

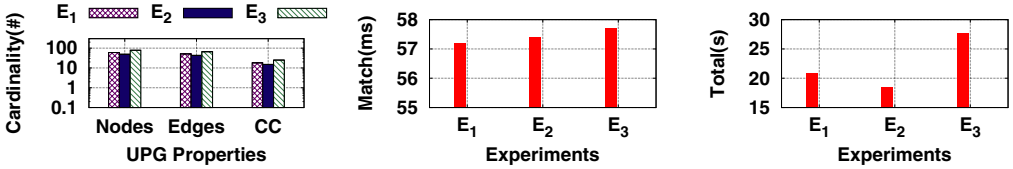
The size and types of log files generated during these experiments are presented in Fig. 4a. The results show that the error logs generated during E_2 are more significant than E_3. In contrary, the access log generated by E_3 is much greater than

[5]https://www.docker.com/ (Accessed: March 23, 2024)
[6]https://docs.docker.com/engine/swarm/ (Accessed: March 23, 2024)
[7]https://hub.docker.com/ (Accessed: March 23, 2024)



(a) Log File Size (b) Phase wise execution time (c) Phase wise Memory utilization

Fig. 4: Runtime Analysis – The Y-axis in (a) is in logarithmic scale



(a) UPG Parameters (b) RegEx Identification Time (c) UPG Construction time

Fig. 5: Runtime Analysis – The Y-axis in (a) is in logarithmic scale

the rest of the two scenarios. However, the audit log generated by E_3 is greater than the other two. The log file size depends on a user's actions while browsing the web service.

Next, we present the phase-wise time consumption analysis in Fig. 4b. Here we observe that the time required to parse the log files, merging them to create an ASCL and generation of UPG during provenance builder (shown in Fig. 5c) is directly related to the total amount of logs generated during the experiments which take approximately 600ms-900ms to complete the runtime processing. After contextualizing the log files, the system administrator provided suspicious log messages converted into an equivalent regular expression (RegEx) to avoid the values of the variables. This generated RegEx string is searched across the LMS-CFG which takes only a few milliseconds (as shown in Fig. 5b). We also provide the memory consumption during the individual phases as presented in Fig. 4c which suggests that the memory footprint is significantly lower for the modules of the runtime engine (in between 200B to 250B).

We observe that depending on the scenarios, the nature of the UPG is different, as shown in Fig. 5a. We find that E_3 has a significantly higher amount of nodes and edges than the rest of the two cases. The number of connected components in the UPG is considerably higher for E_3. Each connected component in the graph represents an execution unit of a process, which helps resolve the dependency explosion problem. Only the related events of a process form a connected component. More number of connected components implies that the graph is more densely partitioned.

VI. Analysis

This section discusses the effectiveness analysis of `DisProTrack`. In this context, we consider an adversarial scenario. The static analyzer can easily detect an adversary who can modify the application's source code. However, an adversary accessing the runtime platform can evade the static analysis and may issue malicious operations during code execution. Therefore, in this section, we primarily focus on detecting runtime adversaries. We have simulated multiple attack scenarios by considering different adversary models. We next discuss one of them.

A. Adversarial model & Attack Scenario

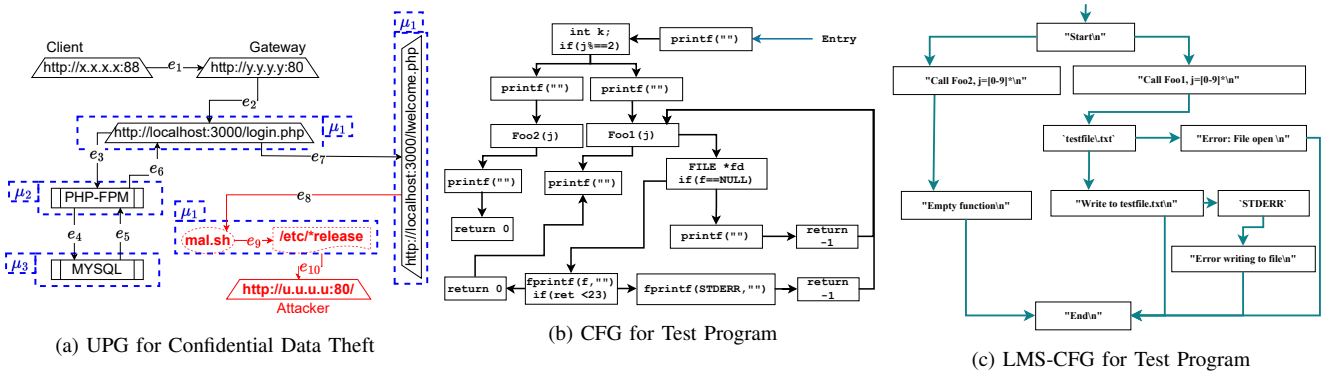
We assume that an adversary can somehow bypass the authorization mechanisms and gain access to one/more application container(s) except the runtime engine container without being detected. In the compromised container(s), the adversary can add, modify, and/or execute scripts and deploy webpages. However, We assume that the logs and audit rules are part of the trusted computing base (TCB). Moreover, the communication between the compromised container and the runtime engine can not be adulterated.

Using this adversarial model, the attacker may perform different types of malicious activities. However, here we present the case study in light of *"confidential data theft"* attack[8] where the attacker attempts to insert a malicious script to steal the confidential information from the compromised system. This script can be triggered during execution time. We simulated the attack on top of our web service application described in Section V-B2 by placing a malicious script named *"mal.sh"* in `PHP-FPM` container. This script is executed when an authorized entity login to the website after successful authentication and clicks on a masked link on the welcome webpage. Once the script gets triggered in the background, alongside normal execution, it tries to access and read a sensitive file on the server and forward them to an attacker's server IP address.

B. Provenance Builder for Attack Detection

Let us understand how this attack can be detected using `DisProTrack`. The UPG constructed by `DisProTrack` for above attack scenario is presented in Fig. 6a. To ensure

[8]https://bit.ly/trendmicro_shading_light (Accessed: March 23, 2024)



(a) UPG for Confidential Data Theft

(b) CFG for Test Program

(c) LMS-CFG for Test Program

Fig. 6: A PoC Case Study to Analyze the Accuracy of DisProTrack

readability, we have omitted a few UPG metadata and masked IP addresses. Using the relative event sequence numbers/edge identifier, the sequence of events can be traced in order. From the particular UPG instance, we can find that a client with IP address $x.x.x.x$ is connected to the service via port 88. The client takes the normal authentication route (from e_1 to e_7) to reach the `welcome.php` page. After that, the `mal.sh` script is triggered which results in collection of data from `/etc/*release` directory and forward it to $u.u.u.u$ (from e_8 to e_{10}). This step is a deviation from the standard behavior; therefore, the system administrator must intervene in this case and take some preventive action (e.g., suspend user access, block IP, etc.).

```c
#include<unistd.h>
#include <string.h>
#include <stdio.h>
#include <stdlib.h>
#define Foo2(int j) ({printf("Empty function\n");})

int Foo1(int j){
    FILE *f = fopen("testfile.txt", "a");
    if (f == NULL) {
        printf("Error: File open\n");return -1;
    }
    if (fprintf(f, "Write to testfile.txt\n") < 23) {
        fprintf(stderr, "Error writing to file\n");
        return -1;
    }
    return(0);
}
/***********************/
int main(void){
    printf("Start\n");int j = rand();
    if(j%2==0){
      printf("Call Foo1(), j = %d\n", j);Foo1(j);
    }else{
      printf("Call Foo2(), j = %d\n", j);Foo2(j);
    }
    printf("End\n");
    return 0;
}
```

Listing 1: PoC Program for Accuracy Analysis

C. Accuracy analysis of DisProTrack

During our development of DisProTrack and experimentation with several other attack scenarios, we observed that the detection of attack depends on the accuracy of LMS-CFG construction from CFG in the static analyzer. Although we have presented the LMS identification results in Fig. 3c for different applications, the lack of gold standard values restricts us from claiming the accuracy of the proposed Algorithm 1. Therefore, to justify the accuracy of the static analyzer, we present a small sample proof-of-concept (PoC) program (Listing 1) here. The sample program presents two function calls depending on a random number generated. The functions can either generate a log message in the STDERR console or in a log file. As the program is simplistic in nature, it is easy to ascertain the accuracy of the generated LMS-CFG from the framework presented in Fig. 6c. For comparison purposes, we present the corresponding CFG in Fig. 6b, which is also obtained from the static analyzer module. From these two figures, we observe that both the figures have 8 listed LMSes and two file handles. The causal paths among the nodes are also verified to ensure that all the paths are covered in the corresponding LMS-CFG.

VII. Conclusion

This paper developed a non-invasive causality analysis framework, called DisProTrack, for provenance tracking over distributed serverless applications. The proposed framework is capable of adversarial attack analysis by identifying the root causes effectively. DisProTrack can be deployed on top of the SLC as a micro-service and has the virtue of being lightweight and provides results within 0.5 minutes. A PoC analysis of DisProTrack also shows its efficiency and efficacy in detecting attack instances for an SLC application.

A critical aspect of DisProTrack is that it uses a heuristic to identify the execution units by matching the CFGs generated from the micro-service binaries with the runtime application and system logs based on their temporal execution patterns. Thus, the framework might wrongly identify the execution units if the underlying servers running the micro-services are not weakly time synchronized (time drift within a small threshold). Nevertheless, this condition rarely occurs in a typical distributed SLC platform with multiple micro-services interacting with each other. Further, DisProTrack can be deployed as an additional micro-service along with the other application micro-services running over the servers, making it robust to be applied for a wide range of production-grade serverless application scenarios.

The authors have provided public access to their code and/or data at https://github.com/usatpath01/DisProTrack.

REFERENCES

[1] K. Kuusinen, V. Balakumar, S. C. Jepsen, S. H. Larsen, T. A. Lemqvist, A. Muric, A. Nielsen, and O. Vestergaard, "A large agile organization on its journey towards devops," in *Euromicro Conference on Software Engineering and Advanced Applications (SEAA 2018)*, 2018, pp. 60–63.

[2] K. Ojo-Gonzalez, R. Prosper-Heredia, L. Dominguez-Quintero, and M. Vargas-Lombardo, "A model devops framework for saas in the cloud," in *Advances and Applications in Computer Science, Electronics and Industrial Engineering*, 2021, pp. 37–51.

[3] F. Alder, N. Asokan, A. Kurnikov, A. Paverd, and M. Steiner, "S-faas: Trustworthy and accountable function-as-a-service using intel sgx," in *ACM SIGSAC Conference on Cloud Computing Security Workshop*, 2019, pp. 185–199.

[4] K. S.-P. Chang and S. J. Fink, "Visualizing serverless cloud application logs for program understanding," in *IEEE Symposium on Visual Languages and Human-Centric Computing (VL/HCC 2017)*, 2017, pp. 261–265.

[5] J. Schleier-Smith, V. Sreekanti, A. Khandelwal, J. Carreira, N. J. Yadwadkar, R. A. Popa, J. E. Gonzalez, I. Stoica, and D. A. Patterson, "What serverless computing is and should become: The next phase of cloud computing," *Communications of the ACM*, pp. 76–84, 2021.

[6] D. Taibi, J. Spillner, and K. Wawruch, "Serverless computing-where are we now, and where are we heading?" *IEEE Software*, pp. 25–31, 2020.

[7] L. Gu, D. Zeng, J. Hu, H. Jin, S. Guo, and A. Y. Zomaya, "Exploring layered container structure for cost efficient microservice deployment," in *IEEE Conference on Computer Communications (IEEE INFOCOM 2021)*, 2021, pp. 1–9.

[8] "Aws x-ray," https://aws.amazon.com/xray/.

[9] "Google cloud - viewing monitored metrics," https://cloud.google.com/functions/docs/monitoring/metrics/.

[10] "Microsoft azure monitor," https://cloud.google.com/functions/docs/monitoring/metrics/.

[11] "Iopipe - monitor serverless applications," https://www.iopipe.com/.

[12] "Dashbird - monitor serverless applications," https://dashbird.io/.

[13] "Thundra - monitor serverless applications," https://www.thundra.io/.

[14] "Epsagon - monitor serverless applications," https://epsagon.com/.

[15] S. Zawoad, R. Hasan, and K. Islam, "Secprov: Trustworthy and efficient provenance management in the cloud," in *IEEE Conference on Computer Communications (IEEE INFOCOM 2018)*, 2018, pp. 1241–1249.

[16] W. U. Hassan, M. A. Noureddine, P. Datta, and A. Bates, "OmegaLog: High-fidelity attack investigation via transparent multi-layer log analysis," in *The Network and Distributed System Security Symposium (NDSS 2020)*, 2020.

[17] W. U. Hassan, A. Bates, and D. Marino, "Tactical provenance analysis for endpoint detection and response systems," in *IEEE Symposium on Security and Privacy (SP 2020)*. IEEE, 2020, pp. 1172–1189.

[18] M. M. Anjum, S. Iqbal, and B. Hamelin, "ANUBIS: a provenance graph-based framework for advanced persistent threat detection," in *ACM/SIGAPP Symposium on Applied Computing*, 2022, pp. 1684–1693.

[19] H. Irshad, G. Ciocarlie, A. Gehani, V. Yegneswaran, K. H. Lee, J. Patel, S. Jha, Y. Kwon, D. Xu, and X. Zhang, "Trace: Enterprise-wide provenance tracking for real-time apt detection," *IEEE Transactions on Information Forensics and Security*, vol. 16, pp. 4363–4376, 2021.

[20] L. Yu, S. Ma, Z. Zhang, G. Tao, X. Zhang, D. Xu, V. E. Urias, H. W. Lin, G. F. Ciocarlie, V. Yegneswaran *et al.*, "ALchemist: Fusing application and audit logs for precise attack provenance without instrumentation." in *The Network and Distributed System Security Symposium (NDSS 2021)*, 2021.

[21] A. Bates, D. J. Tian, K. R. Butler, and T. Moyer, "Trustworthy {Whole-System} provenance for the linux kernel," in *USENIX Security Symposium (USENIX Security 2015)*, 2015, pp. 319–334.

[22] J. Tavori and H. Levy, "Tornadoes in the cloud: Worst-case attacks on distributed resources systems," in *IEEE Conference on Computer Communications (IEEE INFOCOM 2021)*, 2021, pp. 1–10.

[23] Q. Wang, W. U. Hassan, D. Li, K. Jee, X. Yu, K. Zou, J. Rhee, Z. Chen, W. Cheng, C. A. Gunter *et al.*, "You are what you do: Hunting stealthy malware via data provenance analysis." in *The Network and Distributed System Security Symposium (NDSS 2020)*, 2020.

[24] P. Datta, I. Polinsky, M. A. Inam, A. Bates, and W. Enck, "ALASTOR: Reconstructing the provenance of serverless intrusions," in *USENIX Security Symposium (USENIX Security 2022)*, 2022.

[25] "Linux fuse," https://man7.org/linux/man-pages/man4/fuse.4.html.

[26] C. Schaufler, "Lsm: Stacking for major security modules," https://lwn.net/Articles/697259, 2016.

[27] T. F. J.-M. Pasquier, J. Singh, D. Eyers, and J. Bacon, "Camflow: Managed data-sharing for cloud services," *IEEE Transactions on Cloud Computing*, vol. 5, no. 3, pp. 472–484, 2017.

[28] A. Gehani and D. Tariq, "Spade: Support for prmiscovenance auditing in distributed environments," in *ACM/IFIP/USENIX International Conference on Distributed Systems Platforms and Open Distributed Processing*. Springer, 2012, pp. 101–120.

[29] S. Ma, K. H. Lee, C. H. Kim, J. Rhee, X. Zhang, and D. Xu, "Accurate, low cost and instrumentation-free security audit logging for windows," in *Annual Computer Security Applications Conference (ACSAC 2015)*, 2015, pp. 401–410.

[30] M. N. Hossain, S. M. Milajerdi, J. Wang, B. Eshete, R. Gjomemo, R. Sekar, S. Stoller, and V. Venkatakrishnan, "SLEUTH: Real-time attack scenario reconstruction from COTS audit data," in *USENIX Security Symposium (USENIX Security 2017)*, 2017, pp. 487–504.

[31] Y. Xie, D. Feng, Y. Hu, Y. Li, S. Sample, and D. Long, "Pagoda: A hybrid approach to enable efficient real-time provenance based intrusion detection in big data environments," *IEEE Transactions on Dependable and Secure Computing*, vol. 17, no. 6, pp. 1283–1296, 2018.

[32] S. M. Milajerdi, B. Eshete, R. Gjomemo, and V. Venkatakrishnan, "Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting," in *ACM SIGSAC Conference on Computer and Communications Security*, 2019, p. 1795–1812.

[33] S. M. Milajerdi, R. Gjomemo, B. Eshete, R. Sekar, and V. Venkatakrishnan, "Holmes: real-time apt detection through correlation of suspicious information flows," in *IEEE Symposium on Security and Privacy (SP 2019)*. IEEE, 2019, pp. 1137–1152.

[34] J. Zeng, Z. L. Chua, Y. Chen, K. Ji, Z. Liang, and J. Mao, "Watson: Abstracting behaviors from audit logs via aggregation of contextual semantics," in *The Network and Distributed System Security Symposium (NDSS 2021)*, 2021.

[35] S. Ma, J. Zhai, F. Wang, K. H. Lee, X. Zhang, and D. Xu, "MPI: Multiple perspective attack investigation with semantic aware execution partitioning," in *USENIX Security Symposium (USENIX Security 2017)*, 2017, pp. 1111–1128.

[36] K. H. Lee, X. Zhang, and D. Xu, "High accuracy attack provenance via binary-based execution partition." in *The Network and Distributed System Security Symposium (NDSS 2013)*, vol. 2, 2013, p. 4.

[37] ——, "Loggc: Garbage collecting audit log," in *ACM conference on Computer & communications security (SIGSAC 2013)*, 2013, pp. 1005–1016.

[38] S. Ma, X. Zhang, and D. Xu, "Protracer: Towards practical provenance tracing by alternating between logging and tainting." in *The Network and Distributed System Security Symposium (NDSS 2016)*, vol. 2, 2016, p. 4.

[39] R. Yang, S. Ma, H. Xu, X. Zhang, and Y. Chen, "UIScope: Accurate, instrumentation-free, and visible attack investigation for gui applications." in *The Network and Distributed System Security Symposium (NDSS 2020)*, 2020.

[40] R. Fonseca, G. Porter, R. H. Katz, and S. Shenker, "X-Trace: A pervasive network tracing framework," in *USENIX Symposium on Networked Systems Design & Implementation (NSDI 2007)*, 2007.

[41] J. Mace, R. Roelke, and R. Fonseca, "Pivot tracing: Dynamic causal monitoring for distributed systems," in *Symposium on Operating Systems Principles (SOSP 2015)*, 2015, pp. 378–393.

[42] (2007, Mar) auditd(8) - linux manual page. [Online]. Available: https://man7.org/linux/man-pages/man8/auditd.8.html

[43] Y. Shoshitaishvili, R. Wang, C. Salls, N. Stephens, M. Polino, A. Dutcher, J. Grosen, S. Feng, C. Hauser, C. Kruegel, and G. Vigna, "SoK: (state of) the art of war: Offensive techniques in binary analysis," *IEEE Symposium on Security and Privacy (SP 2016)*, 2016.

[44] "Angr," https://angr.io/.

MADE: An MEC Supported Platform Independent and Version Agnostic Framework for Mobile Application Deployment

Arpit Tripathi
IIT Hyderabad & IDRBT Hyderabad, India
Email : cs20resch1006@iith.ac.in

Abhishek Thakur
IDRBT Hyderabad, India
Email : abhishekt@idrbt.ac.in

Abhinandan S. Prasad
IIT Ropar, India
Email : abhinandansp@iitrpr.ac.in

Bheemarjuna Reddy Tamma
IIT Hyderabad, India
Email : tbr@iith.ac.in

Subhrendu Chattopadhyay
IDRBT Hyderabad, India
Email : subhrendu@idrbt.ac.in

***Abstract*—Mobile applications are increasingly preferred by business organizations to engage with their customers. However, traditional mobile application deployments face challenges to handle diversity in devices like model release dates, OS support, storage, computing, etc. These challenges are more prevalent in developing economies due to the higher costs of purchase/upgrades, leading to the resale and redistribution of older mobile devices. In addition, the end users may not follow the best practices for cyber security hygiene. This work presents a network-enabled Multi-access Edge Computing based novel framework *MADE*, for the mobile app ecosystem that is platform-independent, release-agnostic, and secure. *MADE* reduces compute and storage requirements on the user device by offloading the mobile application's functionalities to the edge. The characteristics of the *MADE* framework are evaluated through multiple experiments in a 5G test environment across diverse applications. Analysis of top mobile banking applications from the authors' geography demonstrates that *MADE* is platform-independent, release-agnostic, and well-suited for transaction-based applications.**

***Index Terms*—Multi access edge computing, mobile application development, cross platform, resource management, virtual user interfaces**

I. Introduction

Business organizations (Org) often utilize mobile applications to (a) profile user behavior in a fine grained manner, (b) improve the user experiences [1], (c) utilize device-specific features like camera, microphone, etc. However, maintaining a mobile application, particularly deploying a mobile app using traditional [2] approach is a challenging task for organizations. This paper explores the caveats of the traditional approach to mobile app deployment and proposes improvements through an alternative framework.

The traditional approach [2] for mobile app deployment utilizes non-agile iterative development life-cycle where major web apps adopt continuous integration and continuous delivery/continuous deployment (CI/CD) based approach [3]. The rationale behind traditional non-agile deployment of mobile app is due to the device and platform dependency of the apps. Studies reveal that the diversity of hardware platforms and OS versions across available end-user devices (see Figs. 1 and 2) and mobile phones affect the user experience [4]. Additionally, app developers must consider compatibility with older, resource-constrained mobile phones. This is especially important in developing countries like India, where a significant portion of the population lives in low-income areas[1]. Therefore, apps targeted for mass must optimize battery consumption [5], device lifetime [6], etc. Due to these factors every version of app requires thorough testing using simulators and real devices prior to deployment, adding extra overhead to app maintainers. On the other hand, using older hardware and OS versions can compromise the security, which is well studied in the literature alongside the app's usability [7]. These deployment-related challenges result in app incompatibility for users and lost opportunities, especially for financial transaction-providing services. As a result, app maintainers frequently update and patch their apps (see Fig. 3) to support backward compatibility for multiple versions which leads to complex designs and an increase in operational expenditure (OPEX).

Recent advancements in Multi-access Edge Cloud (MEC) technologies [8], which provides platform-as-a-service (PaaS), addresses the app deployment issues by reducing the resource consumption [9] of the end user devices. Additionally, it enables developers to incorporate CI/CD by rapidly converting mobile apps into microservices. However, the migration to MEC eco-system may increase capital expenditure (CAPEX) and introduce integration complexity. This can be avoided by employing a framework that achieves the following design goals:

① **Platform Independence:** The framework should ensure

979-8-3503-9172-5/24/$31.00 ©2024 IEEE

[1]https://data.worldbank.org/?locations=XM-XN-XT/

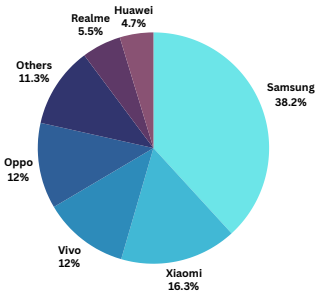


Fig. 1: User %-tage -for Android versions

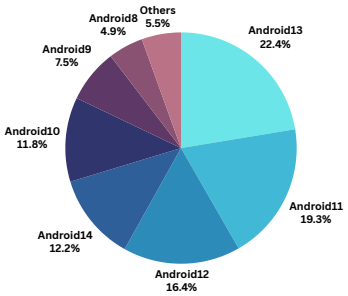


Fig. 2: Market share of Android mobile manufacturers in India

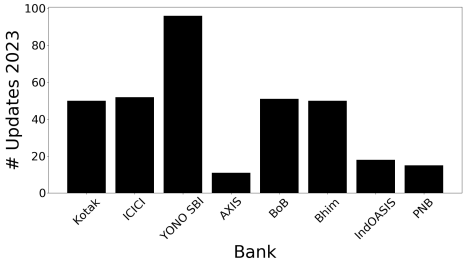


Fig. 3: Mobile app updates in 2023

that the services of the mobile apps work seamlessly, irrespective of the end-user devices or OS version, to the maximum extent feasible.

② **Version Agnostic:** The framework should reduce the app version management overhead to achieve backward compatibility.

③ **Minimal Resource Consumption:** The resource consumption of the end user device should be reduced.

④ **Security Hardened:** As the MEC based framework introduces a new attack surface, the app deployment framework should be at least as secure as the *Traditional* app deployment ecosystem.

⑤ **Rapid Deployment and Adaptable:** The framework should allow CI/CD compatibility without extra overhead which can enable rapid deployment of new features. Moreover, the framework should support the existing mobile apps without any code augmentation.

In this work, we propose **M**obile **A**pp **D**eployment framework in an **E**dge enabled system (*MADE*) which addresses these design goals. The major contributions of this work are:

(a) A comprehensive analysis of mobile banking applications and the range of devices and OS versions they support.

(b) Proof-of-concept (POC) implementation using a MEC enabled physical 5G testbed. The POC implementation is useful to assess the feasibility and detailed requirements of the proposed framework.

(c) Exhaustive experimental evaluation to showcase the capabilities of the proposed framework. We empirically justify the potential of the solution to circumvent the issues in the existing eco-system. Our findings indicate that *MADE* reduces resource consumption for the end-user by marginally increasing the resource overhead on MEC. Additionally, it offers a more flexible ecosystem that facilitates hassle-free mobile app deployment.

The rest of the paper is organized as follows. Section II presents the architecture of *MADE*. The implementation challenges and experimental results are described in Section III. Section IV describes how *MADE* addresses the proposed design goals. The related work and conclusions are presented in Section V and Section VI, respectively.

II. Proposed Framework

The architecture of *MADE* comprises four major components: **User Equipment (UE)**s running a Graphical User Interface (GUI)-based app- "GUI Front-end Application" (GFA), **MECs** running the offloaded app features via "Mobile app EmulAtor middle-ware" (MEA), **orchestration for MEC**, and a **cloud service provider** running the "Back End Application" (BEA). The MECs and cloud service provider are managed by the Org Figure 4 shows the architecture of MADE. It has the following main components:

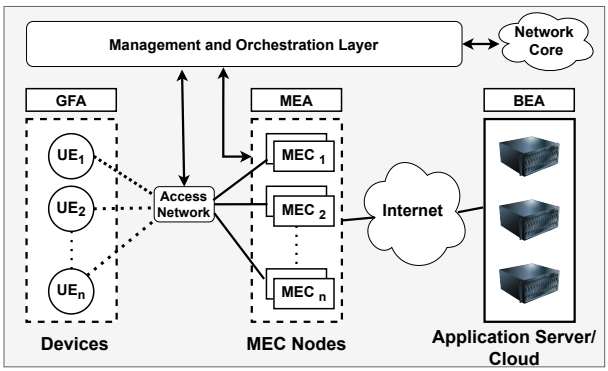


Fig. 4: Proposed MADE Architecture

ⓐ **UE:** UE devices are touch-screen-based mobile devices and use Network Operator's (NOP) services to achieve mobile connectivity. Further, the UEs can run GUI-based client applications to avail the services provided by the Orgs. These devices are also on an unlimited data plan. Therefore, uplink data volume is not a constraint for the user.

ⓑ **MEC:** In the proposed framework, MECs, placed near the edge of the network infrastructure, are capable of providing computing services. The mobile UEs offload their computation tasks to the MEC nodes to reduce resource consumption. The MEC nodes use containers to run the offloaded services.

ⓒ **Cloud Service Provider:** The services provided by the Org are hosted on a secure cloud platform. The choice of cloud (public, private, or hybrid) depends on the Org.

ⓓ **MEC Orchestration:** It manages the MEC nodes [10] to ensure efficient and optimal deployment of apps for various

Orgs. It coordinates with NOP to manage UE mobility and improve user experience.

A. MADE *Framework*

Building upon the previously outlined physical architecture (Fig. 4), *MADE* has three distinct services: a) GFA, b) MEA and c) BEA. GFA acts as a client capable of running a lightweight GUI to interact with and visualize the mobile application hosted in the MEA. In contrast to the typical client-server deployment for mobile applications (apps installed in the UEs), *MADE* deploys the mobile applications in the MEC using a mobile emulator. It provides isolated lightweight sandboxes for security, customization, and manageability. It assumes the presence of end-to-end network connectivity for a transaction-based app ecosystem. The ecosystem is realized using an application flow to access the services provided by the Org.

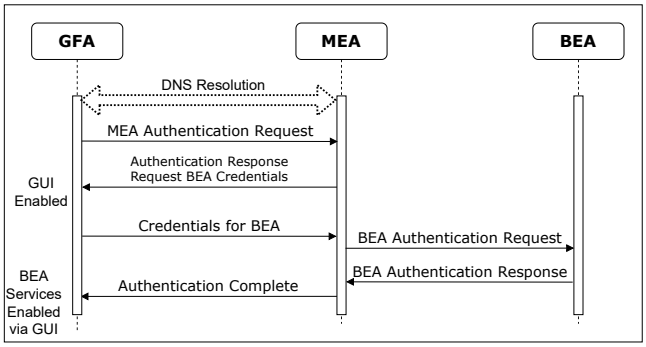


Fig. 5: Sequence diagram for authentication in *MADE*

Fig. 5 illustrates the application flow. Whenever the users want to access the services provided by the Org, they access the GFA, which results in the generation of a `DNS` query. The NOP intercepts the `DNS` request and resolves it with the address of the MEA service. Once the MEA service is discovered, the UE running GFA sends an initial authentication request to the identified MEA. The MEA, based on its internal authentication mechanism, authenticates the UE and requests for BEA credentials. The GFA shares the credentials, which are forwarded to BEA for authentication. As MEA and BEA are both owned by Org, the credentials are as secure as they were in *Traditional* ecosystem. The communication between MEA and BEA remains the same as that of the original Mobile App.

Next, we present a proof-of-concept (PoC) setup to evaluate the *MADE* framework.

III. Proof-of-concept and Evaluation

The experimental evaluation involves five major activities: (i) Creation of a testbed to realize POC for *MADE* along with a custom-built banking application (Custom-App) for controlled experiments, (ii) Evaluating platform independence for multiple devices running diverse Android versions, (iii) Version agnostic behavior of *MADE*, (iv) Overhead evaluation for four popular applications, and (v) Security hardening using a Custom-App.

A. Testbed

*1) **POC hardware setup**:* We have used an in-house 5G testbed to perform the experimentation. This testbed is used only to realize the *MADE* in a cellular deployment physically. The setup includes Open Air Interface (OAI)-based 5G environment[2], along with `ETTUS B210 USRP` as radio unit. We have used 3 Dell Optiplex 5080 workstations, each including an Intel i7-12700 CPU and 16GB of RAM, operating on the Ubuntu 22.04 OS. The first workstation is configured as MEC and 5G Core functionalities using `docker compose` for orchestration. We have deployed MEA using `docker`[3]. To enable GUI, MEA uses `noVNC` server[4] along with Android emulator. The second workstation is configured as a BEA for our Custom-App. We have used 6 physical android devices as UEs and 8 emulated UEs for testing purposes (listed in Table I). The emulated devices are deployed on the third workstation. The physical devices are connected to the cellular network via `ETTUS B210 USRP`. In each UE, we have deployed `Tiger VNC`[5] client app as GFA to interact with the MEA.

*2) **Custom Mobile Banking App**:* We developed a Custom-App for mobile banking to understand back-end server interactions. The mobile app interacts with its customized BEA via `REST` API calls. The BEA is composed of three components("Authentication" and "Balance Enquiry" as business logic microservices and `MySQL` as database).

Fig. 6: POC set-up

*3) **UI automation and metrics collection**:* To maintain the homogeneity of metric computation across all devices, we have used `selenium`[6] to simulate user interactions. This maintains experimental consistency and avoids the unpredictability of human operations. During experiments, we encountered challenges in (a) enabling smooth automation of mobile applications using selenium web plug-in, and (b) collecting fine grained statistics. To overcome the first

[2]https://gitlab.eurecom.fr/oai/cn5g/oai-cn5g-fed
[3]https://www.docker.com/
[4]https://novnc.com/info.html
[5]https://tigervnc.org/
[6]https://www.selenium.dev/

challenge, we used noVNC server in the UE which allows us to access the UE display within a web browser, and control the app using selenium plugin in the browser. To address the second challenge, we have used adb-shell scripts to collect the mobile app performance statistics from UEs.

B. Experiments

Experiments were conducted over the setup mentioned above. Additional steps for the experiments are also captured as applicable.

*1) **Platform Independence**:* In addition to the pixel devices in Table I, we also included an iOS device, running Tiger VNC client as GFA for this experiment. We first attempted to install the mobile banking apps in Table I via adb-shell on all Android UEs in *Traditional* ecosystem. We observed that the latest releases of apps (mostly supporting Android 10 and above) failed to install on older devices, throwing errors similar to what is captured in Listing 1. However, when deployed with *MADE*, the GFA on the devices successfully interacted with MEA (with installed apps), irrespective of the hardware or the OS of the UE. This achieved the platform independence goal as identified in Section I:①.

```
$ adb install -r state-bank.apk
Performing Streamed Install
adb: failed to install state-bank.apk:
  Failure INSTALL_FAILED_NO_MATCHING_ABIS:
    Failed to extract native libraries,res=-113
```

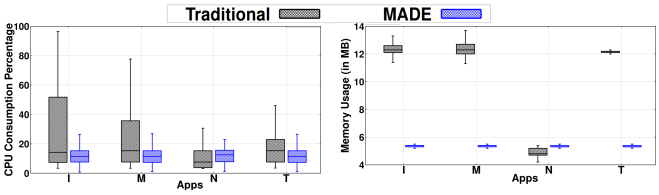
Listing 1: App installation failure

*2) **Version Agnostic**:* The version agnostic design goal, as mentioned in Section I:②, has been tested using twelve prominent mobile banking applications from the Indian banking sector. The initial observations revealed that most Orgs use a standard (full-feature) version of their apps. However, we see instances when Orgs maintain lite and limited-capability versions of their apps. These initiatives target a wider audience and are mostly designed for older Android versions. Compared to the standard app, these apps have limited features. For instance, the Table I shows that “Kotak" maintains two apps, “Kotak 811" app to “open a bank account," "view transaction details," “video verification," etc., whereas the standard “Kotak Mahindra Bank" app, allows additional features like “maintaining credit cards," “fund transfer," “bill payments" etc. Similarly, “State Bank of India" maintains a lite version of their “YONO Mobile Banking" app.

For *MADE* we installed Tiger VNC as GFA on all the GUI-compatible devices. We also installed the standard versions of Mobile apps at MEA. We observed that GFA could successfully access the app installed at the MEA, irrespective of the Android version on UEs.

*3) **Minimal Resource Consumption**:* We installed four popular Android mobile apps to compare the resource consumption and end-to-end behavior in *Traditional* ecosystem with the *MADE* ecosystem. The apps - Instagram (I)[7] for social networking, Maps (M)[8] for navigation, Nettools (N)[9] for network testing and troubleshooting, and Telegram (T)[10] for text messaging, were evaluated. The apps were selected to demonstrate a wide range of possible end-usage. These apps were first tested with the *Traditional* ecosystem. Later, the apps were installed over MEA and tested via GFA. The observations are as follows. CPU consumption and memory usage, were recorded using scripts executed through adb-shell for both scenarios. The plots are presented in Fig. 7a and Fig. 7b. Next, we analyze the observations based on the types of activities recorded for each mobile app, providing a clearer understanding of the findings.

For *(I)Instagram*, we automated user events like scrolling and watching video content, commenting on an existing picture, and uploading an image from the local storage. We instrumented the environment so that the same set of feeds was displayed across runs. Being rich in visual content, Instagram consumed more CPU and memory in the *Traditional* ecosystem when compared to *MADE*. The reduction in *MADE* was because complex processing of the feed was done at the MEA, and only a limited set of visible rectangles as framebuffers[11] were processed by GFA.



(a) CPU Consumption (b) Memory Consumption

Fig. 7: Resource utilization for *Traditional* and *MADE*

For *(M) Maps*, we automated three activities: exploring neighborhoods, downloading local maps, and scrolling around the neighborhood. In the *Traditional* ecosystem, the resource consumption increased with every activity; however, with *MADE*, only the current screen, without any dependency on the rest of the map can be viewed in the GFA. Therefore, *MADE* significantly reduced the resource consumption.

In the case of *(N) Nettools*, we automated activities for ping, traceroute, and network statistics collection. This app is text-based and rarely changes its UI. Therefore, it consumes very little resources while running in *Traditional* ecosystem. The *MADE* ecosystem, which requires screen refresh, increases CPU and memory utilization at UE.

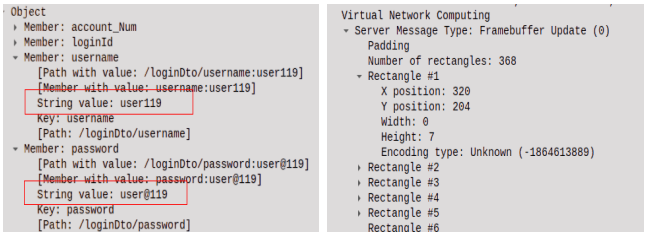
For *(T) Telegram* we automated text entry, sharing a JPEG picture and downloading a large MPEG-4 file (more than 500MB). We observed that resource consumption was more with the *Traditional* ecosystem when compared with *MADE*.

[7]https://www.instagram.com/
[8]https://f-droid.org/en/packages/app.organicmaps/
[9]https://f-droid.org/en/packages/xyz.pisoj.og.nettools/
[10]https://telegram.org/
[11]https://datatracker.ietf.org/doc/html/rfc6143/

	Mobile Devices (Devices- D, Emulators- E)									
Apps	Pixel 2 (E)	Pixel 3(E)	Pixel 3a(E)	Pixel XL(E)	Pixel 5(D)	Pixel 5a(D)	Pixel 5a(D)	Pixel 6(D)	Pixel 6a(E)	Pixel 7(E)
	Android 6	Android 7	Android 8	Android 9	Android 10	Android 11	Android 12	Android 11	Android 12	Android 13
YONO SBI	X	X	X	X	1.23.92	1.23.92	1.23.92	1.23.92	1.23.92	1.23.92
YONO SBI LITE	X	X	5.7.1	5.7.1	5.7.1	5.7.1	5.7.1	5.7.1	5.7.1	5.7.1
BOI	X	X	X	2.0.0	2.0.0	2.0.0	2.0.0	2.0.0	2.0.0	2.0.0
Kotak 811	18.0.1	18.0.1	18.0.1	18.0.1	18.0.1	18.0.1	18.0.1	18.0.1	18.0.1	18.0.1
Kotak Mahindra	X	X	X	X	5.5.4	5.5.4	5.5.4	5.5.4	5.5.4	5.5.4
IDFC	X	X	X	1.31	1.113	1.113	1.113	1.113	1.113	1.113
PNB ONE	X	2.34	2.34	2.34	2.34	2.34	2.34	2.34	2.34	2.34
HDFC	X	X	X	3.2	11.2.3	11.2.3	11.2.3	11.2.3	11.2.3	11.2.3
Axis Mobile	X	X	X	X	X	X	X	9.8	9.8	9.8
VYOM Union Bank	X	X	X	X	X	X	X	8.0.23	8.0.23	8.0.23
BOB	X	X	X	X	3.6.4	3.6.4	3.6.4	3.6.4	3.6.4	3.6.4
IndOASIS	X	X	X	X	X	X	X	2.9.6	2.9.6	2.9.6
IPPB	1.13.2.0	1.13.2.0	1.13.2.0	1.13.2.0	1.13.2.0	1.13.2.0	1.13.2.0	1.13.2.0	1.13.2.0	1.13.2.0

TABLE I: App Compatibility with devices and Android release

To summarize, the *MADE* ecosystem consumed lower resources than the *Traditional* ecosystem for the three apps with interactive multimedia content. *MADE* addressed the minimal resource consumption goal(Section I:③). It ensures users with older and resource-constrained Android smartphones can use various applications. This will also enable the Orgs to expand (or at least retain) their user base without requiring users to upgrade their UEs.



(a) PCAP for Conventional App (b) PCAP for MADE

Fig. 8: Packet capture analysis for password detection

*4) **Security Hardening**:* We conducted experiments using Custom-App on both *Traditional* and *MADE*. To simulate attacks, we captured packets for communication with the UE. Analysis of the capture for *Traditional* shows that, by filtering `HTTP POST` data, the attacker can gain access to the information propagated by the UE (see Fig. 8a). However, when the app is deployed with *MADE*, the client (GFA) only exchanges framebuffer parameters[12] like coordinates, pointer events, and buffer update messages (see Fig. 8b). Extracting the credential from the packet traces in case of *MADE* is significantly difficult. This experiment helped us to demonstrate that the security hardening goal (as discussed in Section I:④), is achieved.

This experiment utilizes a simple baseline security model without enforcing complicated security controls. However, further security enhancements, such as OTP for multi-factor authentication, hardening of MEC runtime, usage of `HTTPS` etc. can be implemented to improve the baseline. Such additional controls will further improve the security posture for *MADE* compared to *Traditional* deployments.

[12]https://datatracker.ietf.org/doc/html/rfc6143/

IV. Discussion

We have experimentally shown that the proposed *MADE* framework addressed ① to ④ in the previous section. This section discusses how *MADE* achieves quick deployment and adaptability design goals ⑤. We also discuss some limitations of *MADE*.

A. *Quick Deployment and Adaptable*

The *MADE* framework allows the Org to run existing apps on MEA. While there is no additional effort for the developer, the end users are masked from the need of constant upgrades on account of new features, security patches etc. The framework ensures adaptability as it can easily accommodate architectural changes for mobile apps.

B. *Overhead of* MADE

There are operational overheads for the Org to setup MEC nodes, maintain user context, manage app releases, and orchestration for MEA. Execution of MEA is an additional cost for the Orgs or users. But, in the case of transaction-based apps, execution-related overhead may be negligible or less significant as the usage patterns are unlikely to peak simultaneously. In [11]–[13], the impact of introducing MEC nodes was studied and evaluated to improve the mobile app performance. Along similar lines, the Orgs can develop a strategy to balance the costs for a number of targeted UEs and the requirements of MEC nodes. Accordingly, the resources can be orchestrated by the NOP.

V. Related Work

In literature, the majority of the works address task offloading via remote display and task splitting in the context of mobile app deployment. These works are mainly motivated by the rapid advancements in the edge-cloud continuum. In [14], an optimization approach was proposed with respect

to the device's battery consumption, wireless bandwidth, and interaction latency in a cloudlet-based architecture. Similar objectives were addressed in [15] by offloading the mobile app features to the cloud while utilizing a GUI-supported user device. In [16], the efficacy of commonly used remote desktop protocols was analyzed using the experimental setup in the context of the user device and mobile app interaction for the cloud. However, these works did not address the implementation-level complexities of deploying mobile apps on the user equipment. On the other hand, the proposed *MADE* framework analyzes the implementation and application level complexities involved in utilizing such frameworks.

In [17], a dynamic task offloading framework for a two-tier cloud environment was presented, introducing complexity due to task offloading decisions. Similarly, [18] presented an estimation technique for various offloading schemes and evaluated in the context of the refactored mobile application. In [19], the task offloading mechanism was addressed using a workload balance algorithm for a drop computing scenario where tasks were offloaded to the neighbor user devices. This added overhead in mobile app development and deployment, which is avoided by *MADE*. Additionally, *MADE* addresses the security issues of collaborative computation in a drop computing platform.

VI. Conclusions and Future Work

The proposed *MADE* framework offloads the app services to the MEC from the UE by executing a GFA at the UE. We setup a POC and analyzed the performance of various apps in a OAI-based 5G MEC deployment. A minimal orchestration mechanism is implemented to run UE specific application instances from the MEC. Our experiments demonstrated that proposed *MADE* achieves the following goals (i) Platform independence, (ii)Version agnostic, (iii) Minimal resource consumption on UE, and (iv) Security hardening. Through analysis, we showed that the *MADE* framework also reduces the development and maintenance efforts for the Org. This work is an initial contribution towards the mobile app development ecosystem using a cloud-edge continuum.

The following activities are planned for *MADE*: (a) Trade-off analysis between usability and security-based on user surveys, subjective experiments, and by integrating *MADE* with a standard benchmarking tool; (b) Expanding the orchestration framework to seamlessly onboard mobile apps and deploy *MADE* in real networks; (c) Integrate the flow of various sensor-related information from GFA to MEA, including a vulnerability analysis for the entire framework; (d) Analysis of the physical setup realized with 5G testbed.

Acknowledgments

This research is supported through funding by the Dept. of Telecommunications, Govt. of India.

References

[1] T. Oliveira, M. Thomas *et al.*, "Mobile payment: Understanding the determinants of customer adoption and intention to recommend the technology," *Computers in human behavior*, vol. 61, pp. 404–414, 2016.

[2] A. Ahmad, K. Li *et al.*, "An empirical study of investigating mobile applications development challenges," *IEEE Access*, vol. 6, pp. 17 711–17 728, 2018.

[3] T. Tegeler, F. Gossen, and B. Steffen, "A model-driven approach to continuous practices for modern cloud-based web applications," in *Proc. of 9th International Conference on Cloud Computing, Data Science & Engineering (Confluence)*, 2019, pp. 1–6.

[4] I. T. Mercado, N. Munaiah, and A. Meneely, "The impact of cross-platform development approaches for mobile applications from the user's perspective," in *Proc. of International Workshop on App Market Analytics*, 2016, pp. 43–49.

[5] H. Jiang, X. Dai *et al.*, "Joint task offloading and resource allocation for energy-constrained mobile edge computing," *IEEE Transactions on Mobile Computing*, vol. 22, no. 7, pp. 4000–4015, 2022.

[6] J. Farman, "Repair and software: updates, obsolescence, and mobile culture's operating systems," *Continent*, vol. 6, no. 1, pp. 20–24, 2017.

[7] D. Smullen, Y. Feng *et al.*, "The best of both worlds: Mitigating trade-offs between accuracy and user burden in capturing mobile app privacy preferences," *Proc. of Privacy Enhancing Technologies*, 2020.

[8] J. Plachy, Z. Becvar *et al.*, "Dynamic allocation of computing and communication resources in multi-access edge computing for mobile users," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 2089–2106, 2021.

[9] J. Liu, J. Ren *et al.*, "Efficient dependent task offloading for multiple applications in mec-cloud system," *IEEE Transactions on Mobile Computing*, vol. 22, no. 4, pp. 2147–2162, 2021.

[10] M. Huang, W. Liu *et al.*, "A cloud–mec collaborative task offloading scheme with service orchestration," *IEEE Internet of Things Journal*, vol. 7, no. 7, pp. 5792–5805, 2019.

[11] R. Fedrizzi, C. E. Costa, and F. Granelli, "A framework to define a measurement-based model of mec nodes in the 5g system," *IEEE Networking Letters*, 2023.

[12] C. Fiandrino, A. B. Pizarro *et al.*, "openleon: An end-to-end emulation platform from the edge data center to the mobile user," *Computer Communications*, vol. 148, pp. 17–26, 2019.

[13] X. Vasilakos, W. Featherstone *et al.*, "Towards zero downtime edge application mobility for ultra-low latency 5g streaming," in *Proc. Of. IEEE Cloud Summit*. IEEE, 2020, pp. 25–32.

[14] M. Satyanarayanan, P. Bahl *et al.*, "The case for vm-based cloudlets in mobile computing," *IEEE pervasive Computing*, vol. 8, no. 4, pp. 14–23, 2009.

[15] P. Simoens, F. De Turck *et al.*, "Remote display solutions for mobile cloud computing," *Computer*, vol. 44, no. 8, pp. 46–53, 2011.

[16] Y. Lin, T. Kämäräinen *et al.*, "Performance evaluation of remote display access for mobile cloud computing," *Computer Communications*, vol. 72, pp. 17–25, 2015.

[17] H. Mazouzi, K. Boussetta, and N. Achir, "Maximizing mobiles energy saving through tasks optimal offloading placement in two-tier cloud: A theoretical and an experimental study," *Computer Communications*, vol. 144, pp. 132–148, 2019.

[18] X. Chen, S. Chen *et al.*, "An adaptive offloading framework for android applications in mobile edge computing," *Science China Information Sciences*, vol. 62, pp. 1–17, 2019.

[19] V. F. Ilie, M. A. Bănică *et al.*, "A computation offloading framework for android devices in a mobile cloud," in *Proc. Of 19th International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*. IEEE, 2023, pp. 98–103.

AutoPAC: Exploring LLMs for Automating Policy to Code Conversion in Business Organizations

Neha Chowdhary
IIT Kharagpur, India
nehachow.cse@kgpian.iitkgp.ac.in

Tanmoy Dutta
IIT Kharagpur, India
duttatanmoy834@gmail.com

Subhrendu Chattopadhyay
IDRBT, Hyderabad, India
subhrendu@idrbt.ac.in

Sandip Chakraborty
IIT Kharagpur, India
sandipc@cse.iitkgp.ac.in

***Abstract*—Managing systems and network policies for large-scale organizations is challenging for business process automation. Although Policy-as-code (PAC) platforms can ease the task of policy management by defining and executing systems and network policies in the form of programmable codes, converting existing organizational policies into PAC-compliant code is not straightforward due to the need for complex dependency resolutions across platforms and applications. On the other hand, policymakers/top management of a business prefer natural language (NL)-based policies that are easy to comprehend. This paper explores large language models (LLMs) to facilitate the automated conversion of NL-based policies to PAC-complaint code. We observe that public LLMs like ChatGPT need thorough multi-round prompt engineering to generate PAC policies. This concerns privacy and security as the organizational policies are sensitive business information. Consequently, we explore using a private and personalized setup, like private LLMs. Notably, we observe that existing personalized LLMs like PrivateGPT fail to understand the system-specific policy semantics. Consequently, we develop a framework called *AutoPAC*, which uses a micro-service architecture coupled with fine-tuned models to generate and validate PAC-complaint policies over a personalized LLM framework. An evaluation with more than 100 test cases indicates that the proposed framework effectively generates and validates PAC policies on the fly.**

***Index Terms*—Policy Management, LLM, PrivateGPT, OPA**

I. Introduction

Large-scale business organizations often need to comply with the policies of regulatory authorities. For example, information technology (IT) operations in banking are often controlled and managed by the policies defined by the central banking authorities of the corresponding country. As an example, the *Reserve Bank of India* (RBI) provides various policies for user authentication for regulatory compliance with cloud-based services[1]. However, in practice, these policies are modified from time to time based on technological upgrades, changes in the organizational frameworks, regulatory compliances, etc. Therefore, managing such policies in a large-scale organization becomes a challenging task, particularly when the IT operations are distributed and scaled across various administrative bodies. Notably, many business organizations have adopted "policy-as-code" (PAC)[2] [1]-based policy management and compliance systems where the policies are stored in the form of programs/code snippets. Among many, PAC provides ambiguity-free interpretation of policies, consistency across multiple systems, low-effort integration in heterogeneous systems, effective version management, and ease of producing compliance reports for internal and external audits.

However, till today, the adoption of PAC is majorly restricted to the IT industries[3], and business organizations face several challenges in migrating from existing systems to PAC[4]. First, existing PAC frameworks use special-purpose languages to represent policies. For example, the *Open Policy Agent* (OPA) [2] framework uses a query language called `REGO` that extends *datalog* (a declarative logic programming language) with policy-specific semantics to represent system-level policies. To utilize the full potential of the PAC frameworks, such PAC-specific languages have a learning curve that the policymakers (often senior management) need to understand and learn. Further, manually translating a policy schema to a PAC-specific language semantics is tedious and often encounters compliance issues, particularly when the policy spans multiple applications or infrastructure. Second, being one of the new technologies, there is a shortage of trained personnel for PAC systems; therefore, it is often a challenge to adopt PAC for enterprise-scale usage. In addition, such PAC frameworks usually do not have a central system to distribute and manage policies across various applications and infrastructures within an organization; consequently, it becomes difficult to figure out what parts of the environment are affected by the policies. Third, policy databases are vital artifacts often kept on secure premises to prevent data and information leakage. Therefore, outsourcing policy enforcement is usually not advisable [3].

To address the above challenges, in this paper, we propose an automation framework to ease the adoption of PAC frameworks for business organizations. The proposed framework, called *AutoPAC*, aims to enable the users to generate PAC-compatible policy (PAC-policy) without any programming experience, thus avoiding the learning curve for PAC-specific languages. Moreover, the framework can avoid the need for trained personnel. For this purpose, we use Generative Pre-trained (GPT) Large Language Models (LLM) to convert the natural language-specific policies into PAC-compatible policies. However, designing such a framework is non-trivial due to the following challenges.

[1]https://learn.microsoft.com/en-us/azure/governance/policy/samples/rbi-itf-banks-2016

[2]https://www.paloaltonetworks.com/cyberpedia/what-is-policy-as-code

[3]https://github.com/open-policy-agent/opa/blob/main/ADOPTERS.md

[4]https://discovery.hgdata.com/product/open-policy-agent

(1) **Organizational Security and Privacy:** The easiest pick to auto-generate PAC-complaint policies would be to use public LLMs, like ChatGPT, Gemini, etc., which have widely been explored for automated code generation [4], [5], [6]. However, such platforms have also been criticized heavily for leaking sensitive information [7], [8]. Further, such public models must be retrained for language and platform-specific semantics, mainly to generate codes from natural language descriptions [8]. Notably, Leaking information about organizational policies may lead to serious security vulnerabilities; therefore, retraining a public model might lead to privacy concerns. Moreover, GPT models require fine training with domain-specific annotated datasets, which are difficult to gather due to the confidentiality of organizational policies.

(2) **Lack of Existing Pre-trained Models:** Existing PAC[5][6][7] platforms often utilize domain-specific descriptive languages to define the policy semantics. Popular GPT platforms such as ChatGPT [9] or Gemini [10] do not perform well for PAC-policy generation tasks as identification of keywords and variables from the set of tokens in a declarative programming language is difficult. Moreover, learning programming semantics is another challenge the existing models face. On the other hand, training models from scratch is a time-consuming operation that may not be affordable for many organizations.

(3) **Verification of Generated PAC-policies:** Although LLMs may be used to generate PAC policies, the generated policies also need to be validated before their deployment. As discussed earlier, validating complex policies across applications and infrastructure is not straightforward. Therefore, policy generation and validation should come hand-in-hand to enable organizations to use the framework reliably and with minimal manual effort.

Considering the above challenges, the proposed framework, *AutoPAC*, utilizes a personalized and private GPT platform to generate PAC-policies from the natural language in a robust, scalable, and privacy-preserved manner. *AutoPAC* is micro-service based; thus, it can achieve rapid deployment using a CI/CD (continuous integration and continuous deployment) pipeline. We first create a dataset of publicly available policies, which has been used to train the model. The framework also requires various pre and post-data cleaning operations to fine-tune the results. We develop a Proof-of-Concept (PoC) implementation, which can be deployed on-premise of an organization to avoid data leakage and requires minimal resource footprints during training and deployment. We use the PoC implementation to perform thorough experiments on fine-tuning, hyper-parameter tuning & model selection. In addition, we develop a unit and integration testing pipeline for comprehensive testing ascertaining the sanity of the generated PAC-policies. Our experimental observations reveal that *AutoPAC* requires less than 2 seconds to generate individual policy. Upon further testing with 116 different test cases, we observed that *AutoPAC* provides almost 94% accuracy in generating the PAC-policies.

The rest of the paper is organized as follows. We first discuss the related literature to highlight existing code generation approaches through LLMs and their limitations in Section II. Section III describes some of our initial experimentation with the existing tools to explain the need for the proposed framework. Section IV and Section V describe the architectural considerations and implementation challenges of the proposed system. In Section VI, we describe the salient features of *AutoPAC* by experimental evaluation. Finally, Section VII concludes the paper.

II. Related Work

This section discusses the related literature that has explored the use of LLMs for code generation and analysis with a highlight on their limitations for PAC-complaint code generation. In this context, we also summarize the works that target to automate policy generation and compliance.

A. Code Generation and Understanding with LLMs

Recent advancements in publicly available LLM tools and models are popular in various use cases, including automated code generation. Existing works [4], [5], [6] have described approaches to generate source codes from prompts. However, automated source-code generation scenarios are beneficial where end users are interested in generating programs that can save time and manual effort. The difficulty in understanding the generated code led to the use of LLMs for code explanation [11], [12] and code understanding [13], [14]. These works proposed using LLMs to develop web browser extensions or plugins to help users understand code snippets. Apart from these, LLMs were also utilized in scenarios where codes were incomplete or incomprehensible for code repair [15]. Notably, these works have primarily analyzed how LLMs can effectively assist code-writing and code-analysis tasks, although concerns about their accuracy in generating platform-specific codes exist. A case study [4] showcases various LLMs having maximum accuracy in terms of functional correctness of the programs as 76.2% with *GPT-4*, 67.1% with *Phind-CodeLlama*, and 64.6% with *WizardCoder-CodeLlama*. Moreover, the existing works have proposed approaches to fine-tune LLMs, particularly for functional languages like C, C++, Python, Java, etc., while to the best of our knowledge, no works consider policy-specific languages like `REGO`.

B. Domain-specific Code Generation

Specialized LLM-based solutions have emerged across various industries as business organizations realized the importance of LLMs in increasing efficiency by avoiding several manual efforts. For instance, LLMs are being used in

[5]https://www.openpolicyagent.org/
[6]https://learn.microsoft.com/en-us/azure/governance/policy/overview
[7]https://www.pulumi.com/docs/using-pulumi/crossguard/get-started/

healthcare industries [16], [17], [18] ranging from day-to-day documentation, sorting feedback of patients, providing an interface for patients to fetch widely available medicinal data to create multiple-choice questions for exams [19] or as self-learning tools [20]. However, we observe that the creation of industry-specific solutions integrating LLMs faces a major hurdle of domain adaptation. Integration can be best utilized when LLMs are trained on the data specific to the domain, thus improving adaptability and accuracy. In contrast to using general-purpose models, we observe from recent literature [21], [22] that domain adaptation is a much more efficient approach for target organizations.

C. LLMs for Policy Generation

Building on the pretext that LLMs are being rapidly adapted for code generation, there emerged the idea to use them for policy generation [23] as well. Available literature emphasizes solutions that have tried to convert user intent into application-specific policies [24]. Robotics at Google [25] also presented using LLM models to produce *robot code*, i.e., planning a sequence of steps from natural language instructions. Among the existing works, a proprietary platform called ARMO [26] is capable of generating REGO-compatible PAC-policies to ensure Kubernetes security. However, we aimed to design a more generalized and useful tool for different business scenarios. Notably, ARMO relies on unit-test-based validation, which may not be feasible for large-scale automation. Moreover, privacy concern significantly hinders adoption of ARMO. Although there exist a few LLM-based natural language to program generators (like *NL2Code* [27], *CodeX* [28], *Code Llama* [29], *StarCoder* [30], *Tabnine*[8], etc.), training domain-specific languages like REGO requires significant resources. Furthermore, LLMs tend to suffer from *hallucinations* [31], [32], [33], [34] where they provide code snippets that are not complete or are incorrect. Notably, code completion tools are available for integration in various IDEs. *GitHub Copilot* [35], powered by OpenAI's *CodeX*, can be used for fetching code suggestions in IDEs as well as seeking chatbot inputs in code. *IntelliCode Compose* [36], a multilingual code completion tool, utilizes *GPT for Code* (*GPT-C*), which is a variant of the GPT-2 model trained from scratch on various source codes written in programming languages like Python, C#, JavaScript and TypeScript. While powerful and robust, this model is mainly based on the code structure and context already available in the code, not on natural languages. Hence, even if fine-tuned for policy-specific coding languages like REGO, its purpose is less suited for PAC-compliant code generation that needs specific semantics based on target infrastructure or applications.

III. Background and Pilot Study

Several NLP-based generative tools like ChatGPT [9], Gemini [10], etc., can generate source codes from the prompt expressing the objectives in natural language. Therefore, we first conduct a pilot study to try the prompt engineering approaches to generate PAC policies for a given scenario. Prompt

(a) Query used with the expected outcome

```
User: Write REGO rules for Global admins and maintainers
      can run any query
Expected: allow {
    object.type == "targeted_query"
    subject.global_role == admin
    action = run
}
allow {
    object.type == "targeted_query"
    subject.global_role == maintainer
    action = run
}
allow {
    object.type == "query"
    subject.global_role == admin
    action = run_new
}
allow {
    object.type == "query"
    subject.global_role == maintainer
    action = run_new
}
```

(b) Response from ChatGPT which has partial similarity with the expected outcome

```
ChatGPT: allow_run_query {
    action == "run_query"
    user.role == "global_admin"
}
        allow_run_query {
    action == "run_query"
    user.role == "maintainer"
}
```

(c) Subsequent prompts used to force ChatGPT to generate a compatible PAC for the target query

```
User (Prompt2): Use object.type==targeted_query or query
      and regenerate
User (Prompt3): Use subject.global_role=admin and action
     == run/run_new and regenerate
```

Fig. 1: Example Query and Responses from *ChatGPT-4*

engineering is essential in such cases as the public LLMs are trained with general-purpose texts. Therefore, they may need platform-specific guidelines through prompt engineering to generate accurate PAC policies. For this pilot study, we consider Role-based Access Control (RBAC) for Amazon-AWS Multi-tenant scenario as a use case[9]. This particular use case contains 2 tenants, each tenant having 5 users. The users access services from one micro-service, which provides "*read*", "*write*," and "*run*" services depending on the tenant category and user role. For the experimental purpose, we have used Open Policy Agent (OPA) [2] to represent the PAC policies. As mentioned earlier, OPA uses REGO [37] as the policy declarative language.

In the first pilot study, we used the ChatGPT V4-based conversational tool to generate a set of equivalent PAC policies. Although ChatGPT or similar public LLMs have privacy concerns, as we discussed before, our objective here is to check (a) how easy it is to generate a PAC policy from

[8]https://www.tabnine.com/

[9]motivated by https://docs.aws.amazon.com/prescriptive-guidance/latest/saas-multitenant-api-access-authorization/opa-abac-rbac-examples.html

natural language descriptions and (b) how good the generated policies are in terms of platform compatibility. Notably, a compatible policy satisfies all the manually generated test cases during OPA testing under various platform constraints. The major challenge, in this case, is the construction of appropriate queries to obtain the compatible PAC policy as a response from the ChatGPT. We observed that, although the ChatGPT-generated responses (see Figure 1b) closely match with the expected outcome (see Figure 1a), it requires fine-tuning to obtain a compatible PAC. Using standard prompt engineering techniques[10], on average, it takes ≈ 4 prompts to get the compatible rules if the queries are invoked under the same topic/chat thread [11]. For example, the ChatGPT response generates a compatible PAC after using the 2 more queries subsequently in the same thread (Listed in Figure 1c). However, one primary concern in using ChatGPT for these types of tasks is related to privacy (as discussed in **Section I:(2)**), which leads to information leakage during this prompt engineering. For example, in the given use case (Figure 1), the system admin needs to disclose the exact fields and variables (e.g., `subject.global_role, action`) used in their system.

To avoid such issues with data privacy, we next explore whether private LLMs like PrivateGPT [38] can work as an alternative. Consequently, in our second pilot study, we use PrivateGPT which utilizes a light-weight pre-trained model (GGML [39]) and can be custom-trained (*known as "ingestion"*) on particular topics. We have deployed the model using a local server with 8 CPU cores and 32GB of RAM as recommended. We have used the AWS documentation (aforementioned) to ingest the model. However, we observed that the results were not always suitable. For example, Figure 2c presents one generated rule for a particular query string as given in Figure 2a. The PrivateGPT-generated PAC is too generic and requires significant training samples to fine-tune it to obtain the desired results. Another observation is that PrivateGPT generates different outputs while querying with the exact prompt multiple times, and the generated outputs are sometimes incoherent[12] as shown in Figure 2c. Moreover, the output generation often takes minutes to respond, and the underlying micro-services crash too frequently, which motivated us to come up with a stable architecture that can provide better responses.

Take Away and Design Goals. The pilot study shows that while public LLMs like ChatGPT need prompt engineering and are thus vulnerable to privacy attacks, private LLMs like PrivateGPT does not provide a robust, consistent output. Consequently, we target to develop a lightweight pluggable framework for domain-specific adaptations over PrivateGPT to achieve automated PAC policy generation. Accordingly, we consider the following design goals for the proposed architecture;

G1. The proposed framework should need minimum domain adaptation and retraining for platform-specific PAC policy generation without a requirement of explicit prompt engineering.

G2. The proposed framework should be scalable and easily deployable following the CI/CD pipeline.

G3. The proposed framework should also be able to validate the generated PAC policies to make them deployment-ready with minimal effort.

We next discuss the details of the proposed framework architecture.

[10]https://www.promptingguide.ai/techniques

[11]https://community.openai.com/t/chat-thread-functionality/60852

[12]The root causes are investigated and explained in Section V.

(a) Query used with the expected outcome

```
User: Write a REGO code for AWS Role-Based Access
      Control (RBAC) of users.
Expected:  allow {
    input.action == "read" or input.action == "write"
    input.repo == "aws"
    input.role == "admin"}
```

(b) Sample Output-1: Output is incorrect but shows some similarity with the expected outcome

```
PrivateGPT:
allow {
    resource : "aws_s3"
    }if (resource.bucket == 'admin-users' and action in
     ['read', 'write'])
```

(c) Sample Output-2: Output consisting of meaningless strings like `o<l,vl,vr,ekr,3>=`

```
PrivateGPT:
allow if { o<l, vl 3>= aws, vr 3>= ekr }
```

Fig. 2: Example Responses from *PrivateGPT*.

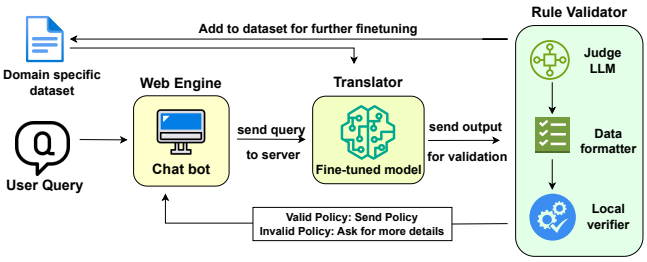


Fig. 3: *AutoPAC*: Proposed Architecture.

IV. Platform Architecture

The *AutoPAC* architecture is sub-divided into 3 major micro-services (as shown in Figure 3): (i) *Web Engine*, (ii) *Translator*, and (iii) *Rule Validator*. The micro-service architecture helps the business administration deploy the platform seamlessly with optimized resource footprints with load-balancing capabilities, thus lowering the response time. The *Web Engine* micro-service is a web-based conversational front-end (chatbot) that provides the user an interface through which they put forth the query in the Natural Language (NL). This component complements the challenge of requiring a complicated learning curve by providing a simple and easily

understandable platform for the user to query even in non-technical language, thereby reducing the need for expertise in policy language to create organizational policies. The query submitted by the user is processed by the *Translator* back-end micro-service, which is connected with the Web Engine over `REST` API. The *Translator* uses a fine-tuned LLM trained explicitly in the PAC-policy language. Upon receiving the user queries in the natural language, the *Translator* generates the corresponding PAC-policies as the output. Since we use LLMs to generate PAC-policies, the output may not always be correct. Therefore, *AutoPAC* uses a separate *Rule Validator* micro-service which validates the generated PAC-policies. The *Rule Validator* is sub-divided into 3 modules as follows: (a) Judge LLM, (b) Data Formatter, and (c) Local Verifier and the working principles of the system are described as follows.

For a system that is supposed to encounter a large number of queries, curating unit test cases for each of them and maintaining test coverage for each generated policy may not be a scalable task. Therefore, the generated PAC-policies are coarsely filtered using an auxiliary LLM termed as *Judge LLM* [40]. This module can verify the output policies depending on various factors such as completeness, legibility, etc. Post coarse filtration, we use fine-grained filtration using subsequent modules. During fine-grained filtration, we use *Data Formatter* module, which formats the *valid* outputs of Judge-LLM using a PAC-policy language-specific "linter"[13]. A linter is a tool used to identify and correct structural errors and stylistic constructs in code. Here, the primary objective of this module is to perform syntactic analysis of PAC-policies, and therefore, it improves the code readability by refactoring them. The formatted PAC-policies are further verified by the user-provided test cases inside a *Local Verifier* module, which contains a set of unit test cases for the given query provided by the user.

The overall architectural design also helps us to future-proof the system. It allows future adaptors to employ a CI/CD pipeline to fine-tune it further. To show the capabilities of this framework, we develop a Proof of Concept (PoC) implementation as discussed next.

V. *AutoPAC*-OPA: PoC Platform Implementation

In order to develop the PoC implementation, we have to custom-train the *AutoPAC* with a particular PAC-policy-specific language. In our implementation, we have identified the popular PAC platform called OPA [2] that uses `REGO` [37] as the policy declarative language. Although OPA does not provide policy enforcement services, it is compatible with multiple open-source projects [41] that can enforce the policies, such as *Kubernetes, Envoy, Express, Terraform, Linux-PAM,* etc. We have created a dataset using publicly available OPA policies to custom-train our framework, as discussed next.

A. Domain-specific Dataset

The created dataset contains `REGO` policies for the OPA framework and their annotations, which constitute a brief description of the rules. To obtain the `REGO` policies, we have selected 19 publicly available repositories (listed in Table I) from `GitHub`. The collected samples mainly contain different types of RBAC and Attribute-based Access Control (ABAC) policies. We have used our customized Python programs to clean the dataset, which includes (a) segregation of comments from the policy code and (b) automated annotations based on the available comments in the program. We used human annotation to label the dataset in a few situations where the comments were unavailable. Finally, we have extracted 1100 such labeled PAC-policies repositories consisting of $\approx$ 50 rules per repository. In total, the dataset contains approx $79,206$ word tokens. During our customized model training phase (described next), we have sub-divided the dataset into 3 divisions (80%, 10%, and 10%) and used them for training, validation, and testing datasets, respectively.

TABLE I: Training Dataset Sourced from Github Repositories

Repository	Link
18F/fleet	https://github.com/18F/fleet
CptOfEvilMinions/fleet	https://github.com/CptOfEvilMinions/fleet
DominusKelvin/fleet	https://github.com/DominusKelvin/fleet
KarlatIwoca/fleet	https://github.com/KarlatIwoca/fleet
TheDyingYAK/splunk_siem	https://github.com/TheDyingYAK/splunk_siem
blazman/fleet	https://github.com/blazman/fleet
empayre/fleet	https://github.com/empayre/fleet
erikng/fleet	https://github.com/erikng/fleet
fleetdm/fleet	https://github.com/fleetdm/fleet
groob/fleetdm-fleet	https://github.com/groob/fleetdm-fleet
kapawit/fleet	https://github.com/kapawit/fleet
kolbeface/fleet	https://github.com/kolbeface/fleet
kyle-humane/fleet	https://github.com/kyle-humane/fleet
lizthegrey/fleet	https://github.com/lizthegrey/fleet
noahtalerman/fleet-1	https://github.com/noahtalerman/fleet-1
stephanmiehe/fleet-1	https://github.com/stephanmiehe/fleet-1
weswhet/fleet	https://github.com/weswhet/fleet
y0zg/fleet	https://github.com/y0zg/fleet
yonnym/fleet	https://github.com/yonnym/fleet

B. Training System

We used a workstation with Ubuntu 20.04.6 and Linux kernel $5.15.0-83-$generic for custom training purposes. The system is equipped with 20 GB RAM, 16 CPU cores, and $12GB$ GPU memory with CUDA 12.2 support. We have used Python-3.10.14 with Conda-24.1.2 for the LLM model training. In this paper, we have adopted the transfer learning approach where we have used a pre-trained model and fine-trained it further with a supervised `REGO`-specific dataset. The functional validation and accuracy of the output is measured by the *Rule Validator* which tests the rules on their syntactical validity and legibility as well as their functional efficiency and provides a score based on the number of test cases passed.

C. Model Selection

In the case of any standard LLM-based transfer learning approach, the choice of the initial LLM model is crucial. In this work, we have tested with 3 existing models as discussed next.

1) ***gpt4all****:* As mentioned in Section III, our initial testing with `gpt4all-ggml` [39] was not satisfactory, and the results generated were not stable most of the time. Upon further investigation, we found that this discrepancy is attributed to the

[13]https://www.openpolicyagent.org/integrations/regal/

inefficient embedding vector representation, which represents the relationship between the word and its contextual meaning of the model. During the training phase, the *vector store*, which refers to a mechanism used for efficiently storing and retrieving vector representations of the data (typically embeddings), is getting updated with the metadata used for training. After the training we observed that various program-specific identifiers were misrecognized such as special symbols like three "==", identifiers like "$vlan$" were incorrectly identified as "3 >=" and "vl" respectively which leads us to the conclusion that the embedding was improper and sometimes arbitrary. Our analysis reveals that the primary reason lies in the Byte-Pair encoding (BPE) scheme used by the model, which often breaks the identifiers used in the source code into sub-strings, which leads to a loss of contextual information that is important for PAC policies. The generated response lacks sanity even after ingesting documents containing multiple `REGO` rules. Based on the intuition that `gpt4all-ggml` belongs to the GPT family, which uses the decoder-only transformer architecture, it lacks deep comprehension of the entire input sequence.

*2) **t5-base**:* Next, we focused on the models that use decoder-encoder transformer architecture to overcome the above issue during model training. Loosely categorizing the PAC-policies as text data, we used *Text-to-Text Transfer Transformer* (`t5-base`) [42] as the base model for the `Translator`. We observed that even though this model uses BPE as used in the previous case, it could extract better contextual information from the submitted dataset of `REGO` codes. However, as the `t5-base` model uses Masked Span Prediction (MSP)[14], it is more sensitive towards declarative languages. Given that the `t5-base` has been predominantly trained on natural language corpus, it becomes less sensitive to the keywords used in programming languages. This can be attributed to the fact that `t5-base`'s pre-training involves *span corruption*, where the tokens of arbitrary lengths are masked, and the model is tasked with predicting them, thus improving context awareness. In the case of source codes, language-specific syntax, data types, control structures, or other structural features of codes need to be considered, which the `t5-base` does consider effectively. However, a few programming language-specific vital tokens (such as colons, parenthesis, identifiers, etc.) were incorrectly identified due to the partial masking of tokens. As an effect, this model disrupts the code structure during training, resulting in the generation of non-compatible `REGO` rules observed in Figure 4.

*3) **codet5-small**:* To overcome the above issue, we finally selected `CodeT5-small`[15] which has been pre-trained in programming languages like `Python`, `C++`, `JAVA`, etc., and was able to learn the program structure of `REGO` including identifiers, keywords, colons, parenthesis, etc. `CodeT5-small` employs a three-step pre-training [43] as follows; (a) Identifier-aware MSP with whole word masking, (b) Identifier Tagging, and (c) Masked Identifier Prediction (MIP). *Identifier-aware MSP* ensures that complete words (especially identifiers) and significant tokens in the code are adequately masked. The approach to treating identifiers as whole units rather than partial tokens prevents disruption of the code structure and allows the model to better understand and predict identifiers in context. On the other hand, *Identifier Tagging* enhances the model's understanding of identifiers by marking them separately from the used variables in an approach similar to syntax highlighting. *Masked Identifier Prediction* instructs the model to precisely mask the identifiers (e.g., variable names, function names) and it is trained to predict these masked identifiers. Specialized masking helps improve the model's ability to anticipate and understand the roles of different identifiers in the code. Our experimental observation reveals that, with the fine-tuned `CodeT5-small` model, the generated PAC-policies codes become acceptably sane most of the time. For example, in Figure 5, the generated `REGO` codes are complete with parenthesis, semicolons, etc. Therefore, for the rest of the PoC implementation, we have continued to use `CodeT5-small` as the selected model.

[14]Tokens are masked arbitrarily over their lengths, often partially, and is used to generate the contextual information

[15]https://huggingface.co/Salesforce/codet5-small

```
User: Write a REGO rule to provide admin access to read and
      write if data.repo==aws
Expected:  allow {
    input.action == "read" or input.action == "write"
    input.repo == "aws"
    input.role == "admin"}
PrivateGPT: tm09in  if dr==aws, c=c and ixrnsx6k. kr.
```

Fig. 4: Sample output using `t5-base` for the given query generates meaningless characters like `tm09in, ixrnsx6k, kr` due to incompatible encoding mechanism.

```
User: Write a REGO rule to provide admin access to read and
      write.
Expected:  allow {
    action == "read" or action == "write"
    repo == "aws"
    role == "admin"}
PrivateGPT: allow {
    global_role == admin
    action == write/read
    repo == aws}
```

Fig. 5: Sample output generated using `CodeT5-small` which closely matches with the expected outcome.

D. Model Fine-Tuning

With the selected model, we started the domain-specific fine-tuning process. During the fine-tuning procedure using standard transfer learning approaches, we have tokenized our dataset (as described in Section V-A) into a maximum of 128 input and output tokens, which were decided based on the highest length of the PAC-policy available in the dataset. We have used `cross-entropy` as the loss function and `decoupled weight-decay Adam optimizer`. The training loss function is defined as the cross-entropy loss over the model's output from the training dataset. In contrast, the validation loss function corresponds to

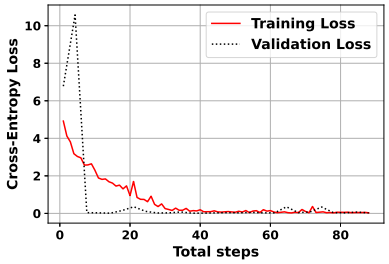


Fig. 6: Cross Entropy Loss during training.

TABLE II: Hyper-parameters used during fine-tuning

Hyper-parameters	Value	Hyper-parameters	Value
Min. Input Tokens	128	Max. Output Tokens	128
Training batch size	8	Validation batch size	2
Epochs	8		

the output from the validation dataset. Our observed training and validation losses per step are shown in Figure 6. We also observed that training over 8 epochs leads to validation loss greater than the training loss. Therefore, we keep 8 epochs for training to avoid over-fitting. From our dataset size of 1100 samples, we choose 80% (880 samples) for training. The batch size 8 indicates that during each iteration, the model processes 8 samples at a time, which amounts to 110 iterations per epoch, totaling 880 iterations over 8 epochs. We choose to use a training batch of size 8 and a validation batch of size 2 based on optimizing training over the available CPU and GPU resources and considering the prevention of overfitting the model to training data. The remaining hyper-parameters were selected based on experimentation and are listed in Table II. Once the training is complete, the generated fine-tuned model is used in our implemented *AutoPAC*.

E. Implementation of Components

We have deployed the individual micro-services using `Docker` to achieve micro-servicification. The details about the components are as follows.

1) Web Engine: The *Web Engine* component uses `ReactJS` to provide the chatbot interfaces to the end users over the web interface. We have observed that this component can behave well with 1 vCPU core and 1GB of RAM. The end-users' queries are forwarded to the Translator, and the response is displayed in the GUI.

2) Translator: The Translator engine has a custom Python-Flask server, allowing other components to access the services via the `REST` API. The incoming user queries to the Flask server are resolved via internal API calls, resulting in the generated `REGO` rules being forwarded to the *Validator* module via the REST API. The post-validated policies are eventually sent as a response to the *Web Engine*. The component is resource-provisioned for optimum performance using 2 CPU cores and 3GB of RAM.

3) Rule Validator: The output is further processed through an on-premise *Rule Validator* to avoid incomplete and irrelevant results generated by the *Translator*. The generated

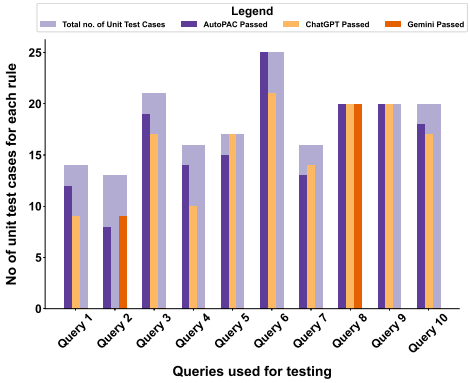


Fig. 7: Unit Testing Comparison of *AutoPAC*-OPA, ChatGPT and Gemini in *Local Verifier*.

`REGO` policies were sent to the publicly available LLMs like ChatGPT [9], which acts as the *Judge LLM*. Although these LLMs themselves can not ensure the guaranteed correctness of the generated PAC-policies, it could still be used to separate out the same PAC-policies from the malformed/incorrect ones. In this case, we emphasize that this testing methodology during fine-tuning does not affect organizational security as the training dataset is publicly available and does not leak sensitive organization-specific policies. Moreover, the user can turn off this verification process and solely rely on the subsequent filtration processes using Linter and unit test cases.

Post Judge-LLM filtration, the generated programs are formatted using the OPA linter module, which checks for the policy syntax and refactors the code into a readable format. In the case of user-provided unit-test cases, the rule is further tested using an OPA unit testing framework, which provides the number of passed test cases along with the generated `REGO` rules.

VI. Experimental Results

We have tested the efficacy of *AutoPAC*-OPA via two mechanisms: (a) Unit test-case validation and (b) using third-party open source LLMs such as ChatGPT-4 and Gemini. For the unit-test-based evaluation, we have shortlisted 10 annotations from our testing dataset (as listed in Table IV) and transformed the annotations into queries. We have manually created $15 - 25$ unit test cases for each query depending on the complexity of the expected `REGO` rules. We have used the *Local Verifier* module to test *AutoPAC*-OPA and compared its accuracy with the existing ChatGPT and Gemini-generated rules (see Table III). In summary, the results (see Figure 7) reveal that, for 6 queries out of the 10 test queries, *AutoPAC*-OPA provided better performance than ChatGPT and Gemini. On the other hand, Gemini and ChatGPT outperform *AutoPAC*-OPA in one (Query-2) and two (Query-5&7) test queries, respectively. For one query (Query-8), *AutoPAC*-OPA, ChatGPT, and Gemini gave equal accuracy.

As generating exhaustive unit testing test cases for a more extensive test dataset is infeasible, we have used ChatGPT-4 to verify the results. We have used Selenium scripts to

TABLE III: Responses Comparison: For the given query, responses generated by each tool are presented along the passed unit test-cases out of 14 test-cases

User: Any user can read/write their own sessions

AutoPAC

```
allow {
object.type == "session"
subject.global_role == admin
action == [read, write][_]
}
```

Test cases passed:(12/14)

Gemini

```
allow {
    input.object.type == "session"
    input.subject.id == input.object.
     user_id
    input.action == "read" ||
    input.action == "write"
}
```

Test cases passed:(0/14)

ChatGPT

```
allow {
    input.action == "read"
    input.object.type == "session"
}
allow {
    input.action == "write"
    input.object.type == "session"
}
```

Test cases passed:(9/14)

TABLE IV: List of Queries Used for Unit Testing

Query Number	Query Description
Query 1	Any user can read/write own session.
Query 2	Any logged in user can read global config.
Query 3	Only global admins and maintainers can read/write packs.
Query 4	Team maintainers can read for appropriate teams.
Query 5	Team admin, maintainer, observer_plus and observer running an observers_can_run query that belongs to their team and there are no target teams.
Query 6	Global admins, maintainers, observers and observer_plus can read all software.
Query 7	Global gitops can write MDM Apple settings.
Query 8	Global admins can read and write Apple devices.
Query 9	If role is observer on any team, can read team details.
Query 10	Only global admins and maintainers can read and write labels.

TABLE V: Accuracy of *AutoPAC*-OPA using ChatGPT

Valid	Invalid	Incomplete
94%	3.4%	2.6%

TABLE VI: Similarity Score with Gold Standard

Sample ID	1	2	3	4
Score	0.95	0.95	0.36	0.36

automate the rule checking for this purpose and observed that, out of 116 generated REGO rules, 94% were approved by ChatGPT. Among others, 3.4% were identified as wrong, and 2.6% resulted in syntactically correct but incomplete rules without any conditions to check for (see Table V). Upon further investigation, we observe that (see Table VI) the cosine similarity score between the Gold Standard (from the annotated Dataset) and the generated REGO rule is very similar for 2 out of 4 failed queries. The generated rules and the corresponding Gold Standard rule are presented in Table VII, which justifies that, even among the failed queries (as evaluated by Judge-LLM), there are a few cases that require minor token alteration to fix the problem.

During the experimentation, we also observed that the responses for the queries require 1.23 ± 0.27 seconds to generate the rules. Moreover, the average CPU and memory utilization of the Translator micro-service is approximately 0.13% and 2.2GB, which justifies the light-weighted nature of the implementation.

TABLE VII: Output Comparison for Failed Queries

AutoPAC-OPA generated	From Dataset	Score
team_role(subject, subject.teams[_].id) == maintainer action == run }	team_role(subject, subject.teams[_].id) == admin action == run_new }	0.95
team_role(subject, subject.teams[_].id) == maintainer action == run }	team_role(subject, subject.teams[_].id) == admin action == run_new }	0.95
team_role(subject, team_id) == [admin, maintainer][_] }	team_role(subject, subject.teams[_].id) == [admin,maintainer][_] action == run_new }	0.36
team_role(subject, team_id) == [admin, maintainer][_] }	team_role(subject, subject.teams[_].id) == [admin,maintainer][_] action == run_new }	0.36

VII. Conclusion and Future Work

In this work, we have proposed *AutoPAC*, a framework to convert natural language-based policies into PAC-policies with the help of LLM. We have created an annotated dataset based on publicly available PAC-policies to fine-tune the LLM. The proposed architecture provides a lightweight, robust, and practical approach to utilizing private LLMs for infrastructure-specific PAC policy generation by combining microservice-based deployment architecture with organization-driven query format processing at its core. As a future extension of this work, we plan to make the setup more scalable. We also plan to test with the more heavyweight models like *CodeX* and *Code Llama* in the Translator architecture and measure if we can seek more accuracy regarding rule structure while optimizing resource consumption. In scenarios where *AutoPAC* gives slightly inaccurate PAC-policies, we plan to conduct a detailed analysis of the similarity using various metrics in future updates. Nevertheless, to the best of our knowledge, *AutoPAC* provides first-of-its-kind private LLM architecture to automate organization and platform-specific PAC policy generation based on natural language queries, which can help organizational governance and IT services automation.

References

[1] M. S. I. Shamim, F. A. Bhuiyan, and A. Rahman, "XI Commandments of Kubernetes Security: A Systematization of Knowledge Related to Kubernetes Security Practices," *IEEE Secure Development*, pp. 58–64, 2020.

[2] OPA, "Open Policy Agent," 2016, [accessed November 13, 2024]. [Online]. Available: https://www.openpolicyagent.org/

[3] Reserve Bank of India, "Master Direction on Outsourcing of Information Technology Services," https://www.rbi.org.in/Scripts/BS_ViewMasDirections.aspx?id=12486, April 2023, [accessed November 13, 2024].

[4] T. Coignion, C. Quinton, and R. Rouvoy, "A Performance Study of LLM-Generated Code on Leetcode," in *International Conference on Evaluation and Assessment in Software Engineering*, 2024.

[5] L. Zhong and Z. Wang, "Can LLM Replace Stack Overflow? A Study on Robustness and Reliability of Large Language Model Code Generation," in *Association for the Advancement of Artificial Intelligence Conference on Artificial Intelligence*, vol. 38, no. 19, 2024, pp. 21 841–21 849.

[6] B. Idrisov and T. Schlippe, "Program Code Generation with Generative AIs," *Algorithms*, vol. 17, no. 2, p. 62, 2024.

[7] Z. Zhang, M. Jia, H.-P. Lee, B. Yao, S. Das, A. Lerner, D. Wang, and T. Li, ""It's a Fair Game", or Is It? Examining How Users Navigate Disclosure Risks and Benefits When Using LLM-Based Conversational Agents," in *ACM Conference on Human Factors in Computing Systems*, 2024, pp. 1–26.

[8] M. L. Siddiq, L. Roney, J. Zhang, and J. C. D. S. Santos, "Quality Assessment of ChatGPT Generated Code and their Use by Developers," in *International Conference on Mining Software Repositories*, 2024, pp. 152–156.

[9] OpenAI, "ChatGPT," https://chat.openai.com/, November 2022, [accessed November 13, 2024].

[10] Google LLC, "Gemini," https://gemini.google.com/app, March 2023, [accessed November 13, 2024].

[11] J. Leinonen, P. Denny, S. MacNeil, S. Sarsa, S. Bernstein, J. Kim, A. Tran, and A. Hellas, "Comparing Code Explanations Created by Students and Large Language Models," in *ACM Innovation and Technology in Computer Science Education*, 2023, pp. 124–130.

[12] P. Vaithilingam, T. Zhang, and E. L. Glassman, "Expectation vs. Experience: Evaluating the Usability of Code Generation Tools Powered by Large Language Models," in *Conference on Human Factors in Computing Systems Extended Abstracts*, 2022, pp. 1–7.

[13] D. Nam, A. Macvean, V. Hellendoorn, B. Vasilescu, and B. Myers, "Using an LLM to Help with Code Understanding," in *International Conference on Software Engineering*, 2024, pp. 1–13.

[14] J. Liu, C. S. Xia, Y. Wang, and L. Zhang, "Is Your Code Generated by ChatGPT Really Correct? Rigorous Evaluation of Large Language Models for Code Generation," *Neural Information Processing Systems*, vol. 36, 2024.

[15] M. Jin, S. Shahriar, M. Tufano, X. Shi, S. Lu, N. Sundaresan, and A. Svyatkovskiy, "Inferfix: End-to-end Program Repair with LLMs," in *ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2023, pp. 1646–1656.

[16] N. Rane, A. Tawde, S. Choudhary, and J. Rane, "Contribution and Performance of ChatGPT and other Large Language Models (LLM) for Scientific and Research Advancements: A Double-Edged Sword," *International Research Journal of Modernization in Engineering Technology and Science*, vol. 5, no. 10, pp. 875–899, 2023.

[17] H. Ali, J. Qadir, T. Alam, M. Househ, and Z. Shah, "ChatGPT and Large Language Models in Healthcare: Opportunities and Risks," in *IEEE International Conference on Artificial Intelligence, Blockchain, and Internet of Things*, 2023, pp. 1–4.

[18] S. Pal, M. Bhattacharya, S. S. Lee, and C. Chakraborty, "A Domain-Specific Next-Generation Large Language Model (LLM) or ChatGPT is Required for Biomedical Engineering and Research," *Annals of Biomedical Engineering*, vol. 52, pp. 451–454, 2024.

[19] Y. Artsi, V. Sorin, E. Konen, B. S. Glicksberg, G. Nadkarni, and E. Klang, "Large Language Models for Generating Medical Examinations: Systematic Review," *BioMed Central Medical Education*, vol. 24, no. 1, p. 354, 2024.

[20] W. Choi, "Assessment of the Capacity of ChatGPT as a Self-Learning Tool in Medical Pharmacology: A Study Using MCQs," *BioMed Central Medical Education*, vol. 23, no. 1, p. 864, 2023.

[21] N. Zhang, Y. Liu, X. Zhao, W. Cheng, R. Bao, R. Zhang, P. Mitra, and H. Chen, "Pruning as a Domain-Specific LLM Extractor," in *Findings of the Association for Computational Linguistics*, 2024, pp. 1417–1428.

[22] Y. Ge, W. Hua, K. Mei, J. Ji, J. Tan, S. Xu, Z. Li, and Y. Zhang, "OpenAGI: When LLM Meets Domain Experts," in *Neural Information Processing Systems Datasets and Benchmarks Track*, 2023.

[23] D. Ferrin, "How LLMs Can Simplify Writing Company Policies and Procedures," 2023, [accessed November 13, 2024]. [Online]. Available: https://projectinggroup.com/how-llms-can-simplify-writing-company-policies-and-procedures/

[24] K. Dzeparoska, J. Lin, A. Tizghadam, and A. Leon-Garcia, "LLM-Based Policy Generation for Intent-Based Management of Applications," in *International Conference on Network and Service Management*, 2023, pp. 1–7.

[25] J. Liang, W. Huang, F. Xia, P. Xu, K. Hausman, B. Ichter, P. Florence, and A. Zeng, "Code as Policies: Language Model Programs for Embodied Control," in *IEEE International Conference on Robotics and Automation*, 2023, pp. 9493–9500.

[26] ARMO, "ARMO and ChatGPT – Create Custom Controls Faster," 2023, [accessed November 13, 2024]. [Online]. Available: https://www.armosec.io/blog/armo-chatgpt-create-custom-controls-faster/

[27] F. F. Xu, B. Vasilescu, and G. Neubig, "In-IDE Code Generation from Natural Language: Promise and Challenges," in *ACM Transactions on Software Engineering and Methodology*, 2021.

[28] M. Chen and J. Tworek, "Evaluating Large Language Models Trained on Code," *IEEE/ACM International Conference on Software Engineering*, 2021.

[29] B. Roziere, J. Gehring, F. Gloeckle, S. Sootla, I. Gat, X. E. Tan, Y. Adi, J. Liu, R. Sauvestre, T. Remez *et al.*, "Code Llama: Open Foundation Models for Code," *International Workshop on Large Language Models for Code*, 2023.

[30] R. Li, L. B. Allal, Y. Zi, N. Muennighoff, D. Kocetkov, C. Mou, M. Marone, C. Akiki, J. Li, J. Chim *et al.*, "StarCoder: May the Source Be With You!" *Transactions on Machine Learning Research*, 2023.

[31] A. Eghbali and M. Pradel, "De-Hallucinator: Mitigating LLM Hallucinations in Code Generation Tasks via Iterative Grounding," *arXiv preprint arXiv:2401.01701*, 2024.

[32] F. Leiser, S. Eckhardt, M. Knaeble, A. Maedche, G. Schwabe, and A. Sunyaev, "From ChatGPT to FactGPT: A Participatory Design Study to Mitigate the Effects of Large Language Model Hallucinations on Users," in *ACM Mensch und Computer*, 2023, pp. 81–90.

[33] A. Martino, M. Iannelli, and C. Truong, "Knowledge Injection to Counter Large Language Model (LLM) Hallucination," in *European Semantic Web Conference*. Springer, 2023, pp. 182–185.

[34] Z. Ji, T. YU, Y. Xu, N. Lee, E. Ishii, and P. Fung, "Towards Mitigating LLM Hallucination via Self Reflection," in *Empirical Methods in Natural Language Processing*, 2023, pp. 1827–1843.

[35] "What is GitHub Copilot?" 2023, [accessed November 13, 2024]. [Online]. Available: https://docs.github.com/en/copilot/about-github-copilot/what-is-github-copilot

[36] A. Svyatkovskiy, S. K. Deng, S. Fu, and N. Sundaresan, "IntelliCode Compose: Code Generation Using Transformer," in *ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1433–1443.

[37] REGO, "Open Policy Agent — Policy Language," 2016, [accessed November 13, 2024]. [Online]. Available: https://www.openpolicyagent.org/docs/latest/policy-language/

[38] "Private GPT," https://hub.docker.com/r/rattydave/privategpt, [accessed November 13, 2024].

[39] R. Marella, "gpt4all-j-ggml," https://huggingface.co/rustformers/gpt4all-j-ggml, 2023, [accessed November 13, 2024].

[40] L. Zheng and W.-L. C. et. al., "Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena," in *International Conference on Neural Information Processing Systems*, vol. 36, 2023, pp. 46 595–46 623.

[41] Open Policy Agent, "OPA Eco-system REST API Integrations," https://www.openpolicyagent.org/ecosystem/rest-api-integration/, April 2023, [accessed November 13, 2024].

[42] H. W. Chung and L. H. et. al., "Scaling Instruction-Finetuned Language Models," in *Journal of Machine Learning Research*, vol. 25, no. 70, 2022, pp. 1–53.

[43] Y. Wang, W. Wang, S. Joty, and S. Hoi, "CodeT5: Identifier-aware Unified Pre-trained Encoder-Decoder Models for Code Understanding and Generation," in *Conference on Empirical Methods in Natural Language Processing*, 2021, pp. 8696–8708.

BeeGuard: Explainability-based Policy Enforcement of eBPF Codes for Cloud-native Environments

Neha Chowdhary
IIT Kharagpur, India
nehachow.cse@kgpian.iitkgp.ac.in

Utkalika Satapathy
IIT Kharagpur, India
utkalika.satapathy01@gmail.com

Theophilus Benson
Carnegie Mellon University, USA
theophib@andrew.cmu.edu

Subhrendu Chattopadhyay
IDRBT, Hyderabad, India
subhrendu@idrbt.ac.in

Palani Kodeswaran
IBM-IRL, Bangalore, India
palani.kodeswaran@in.ibm.com

Sayandeep Sen
IBM-IRL, Bangalore, India
sayandes@in.ibm.com

Sandip Chakraborty
IIT Kharagpur, India
sandipc@cse.iitkgp.ac.in

***Abstract*—eBPF enables loading user space code into the kernel, thereby extending the kernel functionalities in an application-aware manner. This flexibility has led to the widespread adoption of the technology across hyperscalers and enterprises for several use cases, including observability, security, network policy enforcement, etc. In general, the safety of the loaded eBPF programs are ensured through a kernel verifier that performs different syntactic/structural checks to secure the kernel against unwanted crashes. In this paper, we motivate the case that this verifier-based security check, while necessary, is insufficient to ensure that the deployed eBPF code complies with organizational policies. Consequently, we propose *BeeGuard*, a framework to understand program behavior to extract capability lists from eBPF programs. *BeeGuard* introduces a policy compliance layer on top of the existing verifier, and the extracted capability lists of a program are then checked against organizational policies to allow or block loading the programs. Thorough experiments across the most popular open-source eBPF tools show that *BeeGuard* can enforce typical enterprise policies with minimal overhead in terms of loading latency and resource utilization.**

***Index Terms*—eBPF, observability, kernel verifier, policy compliance**

I. Introduction

Observability and efficiency improvement of large-scale distributed systems (such as Serverless platforms [1], [2], hyperscalers [3], content delivery networks (CDNs) [4], etc.) has gained a lot of attention in recent literature [5]–[7]. Among these, several solutions have utilized *Extended Berkeley Packet Filters* (eBPF) [8] to design fine-grained monitoring platforms [9] for cloud-native applications deployed over such large-scale systems. The advantage of eBPF is that it allows developers to introduce custom functionalities into the kernel transparently, which improves efficiency [10]–[12] and flexibility [13], ranging from security, observability [9], [14]–[17], network profiling [18] and policy enforcement [19]–[23], etc. eBPF source codes are compiled into a *machine-independent intermediate representation* (also known as bytecodes) that is further compiled with a Just-in-Time (JIT) compiler at runtime to achieve platform independence. In addition, each eBPF program passes through an in-kernel verifier, which ensures that the program does not initiate kernel panic. Furthermore, eBPF utilizes in-kernel sandboxing, which restricts the eBPF program from directly modifying kernel data structures, thus enhancing the platform's security. To access the kernel information, each eBPF program is associated with triggering events (hook points) that initiate the program's execution. Additionally, eBPF programs use specialized data structures (known as "maps") to enable data exchange across multiple eBPF and userspace programs.

Despite these advantages, using the eBPF eco-system leads to compliance concerns for large regulated organizations. For example, let us consider the scenario where a banking and financial organization utilizing eBPF programs also needs to ensure policy compliance strictly. The task becomes challenging for compliance officers of the banking organization who have minimal understanding of the eBPF eco-system. The primary hurdles are as follows.

① eBPF programs deployed inside kernel enjoy greater privileges, and therefore, runtime risk analysis for the deployed eBPF programs is non-trivial. Given a set of organizational policies, it requires technically skilled auditors with in-depth knowledge of the eBPF eco-system to check if the deployed eBPF programs are actually in compliance with the organizational policies. Notably, reverse engineering of eBPF bytecodes [24] has been proposed, which requires skilled reverse engineers to understand the program behavior.

② Policies for kernel-level programs should be implemented inside the kernel. However, in large-scale organizations, policies are dynamic and change quite often, which is difficult if the policies are directly implemented inside the kernel.

③ Implementing source control has been challenging for eBPF programs. Existing works have relied on the hypervisors [25] and customized device drivers [26] to check the sanity of the deployed eBPF programs. However, hypervisors and device drivers need an additional level of redirection, thus significantly increasing the program load time. On the other hand, implementing a signature scheme to ensure source control requires trusted user space applications [27].

To address the above challenges, in this paper, we propose a framework called *BeeGuard*, which develops an observability framework for the eBPF programs during runtime to allow/deny the programs based on compliance with the

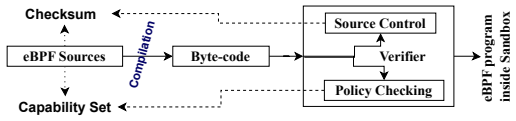


Fig. 1: Conceptual view of *BeeGuard*

organizational policies. We hypothesize that selecting a suitable behavioral model of an eBPF program which can even be understandable by the policy makers/compliance officers having little or no understanding of eBPF construct and accurately capture the key behaviors without instrumenting the source code can address the first challenge mentioned above. Notably, the same behavioral model can also be utilized to represent the policies. To address the second challenge, we propose separating policy management from policy enforcement so that policy compliance becomes easier for naive users. For example, a global policy store where policies can be loaded dynamically and a policy enforcement framework that can continuously police the system is required. Finally, to address the third challenge, we propose an in-kernel source control primitive to ensure the non-compromization of eBPF programs from a trustworthy source without relying on any third-party/user-space application.

The core of our proposed solution works as follows. To express the behavioral model, we use a novel approach of utilizing the "*capability set*" of an eBPF program, which is defined as a list of used hook points, along with maps and header functions used to develop the eBPF program. We use a natural language processing (NLP)-enabled code analysis module, which can extract the capabilities of an eBPF program from its source code. As shown in Figure 1, the extracted capabilities are used by the in-kernel verifier, which checks if the program complies with the organizational policies expressed in terms of capabilities and if the source is from a legitimate source (via source-control primitive). Our implementation requires minor kernel-level modification ($\approx$ 160 LOC) for the in-kernel verifier, which makes it easy to incorporate into mainline kernel releases. Moreover, the other components of *BeeGuard* are deployed as containerized microservices, making it scalable and easy to manage.

We have tested *BeeGuard* using 15 publicly available popular eBPF programs (totaling $\approx$ 500 lines of codes) used for cloud-native development. The experimental results suggest minimal overhead during the program load time ($\approx$ 3ms), very minimal overhead in CPU and memory utilization ($< 1\%$) during program loading, and zero overhead during the eBPF execution time. We have also observed that the enforcement framework can block eBPF programs that do not satisfy the policy specifications. Furthermore, qualitative experiments indicate that the modification of the policies requires minimal effort, while the implementation overhead of the modified policy is negligible ($<$ 10ms). Our qualitative analysis shows that the proposed framework can effectively enforce policy compliance for the deployed eBPF programs in a scalable, robust, and efficient way. In summary, the contributions of this paper are as follows.

1) We develop a novel framework for in-kernel processing of organizational policies and use it during the eBPF program loading to enable or disable a program from loading over the runtime environment.
2) We utilize the program capabilities extracted from the eBPF programs to develop a novel in-kernel verifier extension to check whether the program capabilities match the organization's policies, thus enforcing policy compatibility during eBPF runtime.

II. System Architecture

Trust Model: In designing *BeeGuard*, we make the following trust assumptions. We assume the kernel and eBPF verifier are trusted and un-compromised entities. We also assume that the attacker can compromise any client device and alter the eBPF program sources using a compromised user-space program. For this reason, we can not trust the eBPF loader programs as the loader programs execute at the user space.

System Overview: We design *BeeGuard* as a centralized controller-based architecture to provide flexibility in policy management using two main components: (a) Global eBPF Sentinel (G-sent), and (b) Kernel-space (KSpace) as depicted in Figure 2.

Here, the G-sent is a central component executing inside a secured environment (such as a demilitarized zone (DMZ) [28]), responsible for the overall policy management. The DMZ-based deployment makes G-sent immune to the compromised eBPF programs. On the other hand, the "KSpace" component is part of the eBPF runtime environment and is executed on each device where the eBPF programs are supposed to be deployed. The "KSpace" is responsible for policy enforcement. Following the standard practice, the eBPF source codes are compiled to the bytecode at the user space (USpace) of each device (except DMZ) where they are supposed to be deployed using the runtime environment as shown in Figure 2. Using an untrusted loader eBPF can be loaded into the kernel which internally invokes the eBPF verifier. At this time the "policy fetcher" module at the "KSpace" checks the compliance of the eBPF program with the help of G-sent before actually allowing the eBPF program to execute. The implementation details of *BeeGuard* are as follows.

A. Global eBPF Sentinel (G-sent)

The *Global eBPF Sentinel* (G-sent) provides a single point of management that offers a dashboard to allow easy interaction between the system administrator and *BeeGuard*. This dashboard can be used to introduce eBPF programs into the system. Since this component also stores the organizational policies, it provides ease of augmenting the policies as per the requirements. Although implementation of this module in the kernel space may improve security, it significantly reduces the ease of policy modification. The policies are represented as a set of allowed and blocked *"capabilities"*. In this context, we define a capability as a "sub-action"

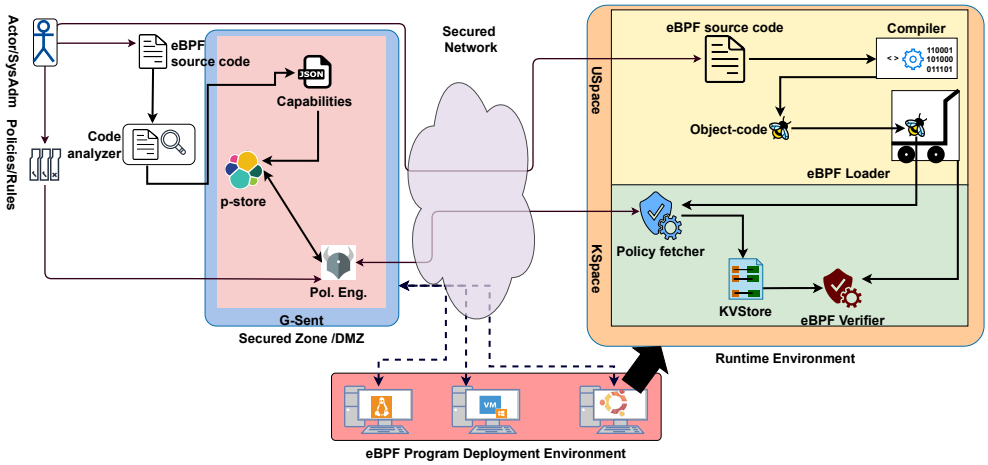


Fig. 2: Proposed architecture of *BeeGuard*

which can be performed by a program. For example, packet header manipulation, system tracing, packet cloning, etc. can be considered capability labels. We assume that an `eBPF` program is represented as a set of capability labels and stored inside the "*Persistent Capability Store*" (p-store) sub-module of `G-sent` (See Section II-C for proof of concept). The stored capabilities are referred to by the "*Policy Engine*" sub-module to ascertain the compliance of the program as per the organizational policies.

1) p-store: The rationale behind using the capabilities from each `eBPF` program is that the set of capabilities together represents the behavioral model of the `eBPF` programs. "p-store" works as a persistent store for the behavioral model so that the policy compliance checking can be expedited by the use of an index to query the capabilities for each function of the `eBPF` program. For our implementation purpose, we have used the *Elasticsearch database* [29], which is a scalable, fast, lightweight, fault-tolerant database. We have used containerized deployment of Elasticsearch to ease the deployment process. The capabilities stored in "p-store" are frequently consulted by the "*Policy Engine*", as discussed next.

2) Policy Engine: The objective of the "*Policy Engine*" is to ascertain if the invoked `eBPF` program violates the system-wide policy specifications. Suppose at least one capability is incompatible with the system, then the "*Policy engine*" can decide not to allow such an `eBPF` program. In this paper, we have primarily focused on the restrictions that should be enforced related to `eBPF` programs. To ease the policy management, we have used Open Policy Agent (OPA) [30], which can store and manage the policies in the form of codes and provides a platform to control the loading or rejection of `eBPF` programs as per the organizational policy. Although the list of policies can be extended significantly, we have implemented three different types of policies for experimental purposes to allow programs with specific capabilities from a trusted source and not having certain capabilities (more details in Section III-B). The decisions taken by the `G-sent` are enforced by the rest of the two components.

B. Kernel-space Component (`KSpace`)

The policy enforcement process for each `eBPF` program is carried out by the `KSpace` component, which controls the permission of the target `eBPF` program, either allowing or blocking its execution. This placement of the enforcement framework in the kernel space is strategic, providing enhanced security and resilience against modifications compared to user-space implementation. In the interaction between the "G-sent" and "KSpace", we have introduced a "Policy Fetcher" module. The overall sequence of execution is as follows.

The `eBPF` source code, once compiled to generate equivalent bytecode, is loaded via a user-space loader program. During the loading, the bytecode is submitted to the in-kernel verifier. The verifier interacts with the `G-sent` module via "Policy Fetcher", which is a customized Loadable Kernel Module (LKM) and is responsible for fetching decisions about the target `eBPF` program and storing it inside the kernel space enforcement component securely. Additionally, the loader program loads the `eBPF` program into the memory and hands it over to the kernel-space "`eBPF` verifier". Since "Policy Fetcher" is developed as an LKM, it can be secured via traditional signature-based mechanisms. The communication between `G-Sent` and the "Policy fetcher" is assumed to be secured using SSL-based encryption.

Additionally, $\approx$ 160 lines of code were added inside the kernel to modify the `BPF_PROG_LOAD` system call. This particular system call is used to load an `eBPF` program and leverages the `bpf_attr` structure to store attributes of the program such as program type, instruction count, etc. as input arguments. This system call is also responsible for the allocation of maps and verification of the `eBPF` program subsequently using the `BPF_CHECK` system call.

Our proposed enforcement logic validates the authenticity of the program, fetches the corresponding *policy decision*, and enforces the decision before sending the `eBPF` program for verification. This proposed code modification is independent of the platform or versions and written with the intent to be

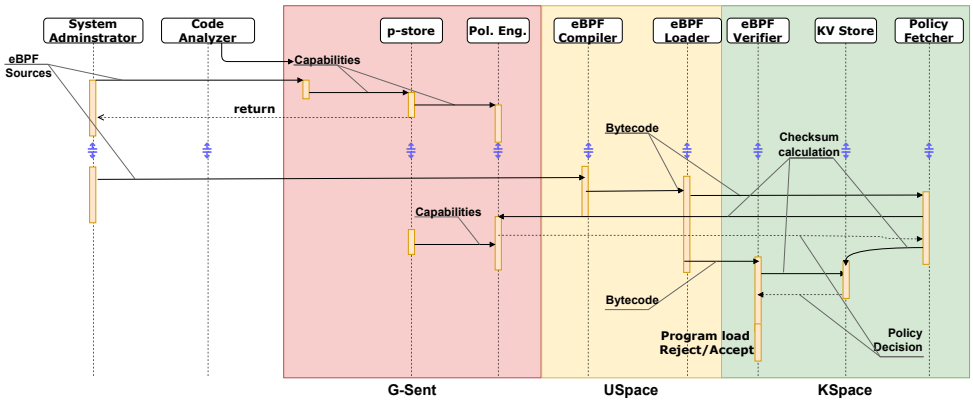


Fig. 3: Sequence diagram for loading of an eBPF program

officially incorporated in mainstream kernel versions in due course. The augmented code performs two specific tasks; (a) checks the program identity, and (b) compares it with the allowed capabilities provided by G-sent.

To calculate the unique identity (ID) we have implemented an in-kernel check-sum-based signature calculation function. The obtained ID is cross-referenced with an in-kernel local Key-Value cache ("KV Store"). The "KV Store" keeps the program ID as a key and the decision variable (allow/block) as a value. The entries in the "KV Store" are updated by the "Policy Fetcher". The "KV Store" avoids performing costly policy fetching operations from the G-sent server each time the programs are loaded. The ID can also be leveraged as signatures to implement the source control mechanism. Based on the decision variable inside the "KV Store" the system call determines whether to proceed by forwarding the eBPF program to the verifier block or to return an error code. The overall interaction between these components is depicted in Figure 3.

During the implementation, we faced the following two major challenges; (a) checksum computation for an eBPF program, (b) interaction with the "G-sent". While attempts have been made to extend the source control mechanism [31] of LKM to the eBPF subsystem, none have been accepted by the community so far due to the two-phase compilation process of eBPF. During the JIT compilation of eBPF programs, the bytecodes are optimized using code relocation techniques and then converted into machine code. This process changes the calculated checksum at the time of loading. Another difficulty in checksum calculation during the program verification is due to the size restriction of the eBPF programs. To avoid the restriction, each eBPF program is subdivided into multiple subroutines before verification. To overcome this challenge, we have utilized eBPF's Instruction Set Architecture (ISA) [32] and devised our custom checksum mechanism from the bpf_insn() structure which can be generated in both userspace and kernel space. Once the checksum is calculated, it is used to query the G-sent about the authenticity and policy compliance of the program. However, consulting G-sent each time before loading a program becomes time-consuming; therefore, we have used the local "KV Store".

C. Code Analyzer

So far, we have assumed that the behavioral model of an eBPF program is available for perusal, which can be achieved in multiple ways through static analysis [33], [34]. It has been observed that traditional approaches like binary analysis [35] provide significant information about the structure of the program but are not suitable for predicting output from the program; hence, modern static analysis approaches rely on neural machine translation (NMT) [36]. Therefore, we have relied on a similar approach to design our *Code Analyzer* module in this work. This module aims to find the list of capability labels of each eBPF program. These capability labels of the program specify the set of actions, for example, manipulating the IP header, adding a VLAN header, dropping packets, etc., that the program could potentially perform at run time. These capability labels are fed into the policy enforcement engine and used when authoring enterprise policies.

The proposed *Code analyzer* uses the insight that the eBPF programs can *only use* well-defined helper functions to read and manipulate the system states. Consequently, we have focused on extracting and leveraging information about eBPF helper functions for extracting the capability labels of the program. To generate the capability labels, the code analyzer leverages a combination of (a) *static code analysis* and (b) *semantic understanding* to understand program/functional intent from an enterprise perspective. When a new eBPF program is submitted to the system, the *Code Analyzer* runs a static analysis pass on the source code to extract the program features such as the set of eBPF maps that each function reads/writes, eBPF helper functions per function calls, etc. to identify program functionality and potential developer intent. To extract semantic capabilities, *BeeGuard* employs a domain-specific NLP pipeline. The tool is pre-trained with a custom-built dataset consisting of a human-annotated dataset[1]. The NLP pipeline leverages a combination of sources such as

[1]https://github.com/eBPFDevSecTools/annotations (Accessed: November 12, 2024)

helper function man pages, program comments, human annotations of eBPF codes, etc. Specifically, our NLP pipeline can identify action words from man pages of the helper functions and programmer comments. This set of actions is then filtered using a curated list of enterprise-sensitive action words that are of typical importance from a policy perspective, such as packet manipulation, packet dropping, system tracing, etc. Additionally, we leverage the comments around the eBPF map definitions to identify potential semantic data types such as IP header, MAC address, cgroup ID, etc., stored in the map.

The combination of map-related helper functions (obtained from static analysis) along with the map data types (inferred from comments) to associate additional capability labels. For instance, a program that *writes* to a map and stores the *source IP address* is assigned the capability label *store_header*. Furthermore, we have created a human annotation dataset describing the functionality of each eBPF function in natural language. Our pipeline extracts action words from these human annotations to extract higher-level capability labels, such as load balancer, port forwarder, firewall, etc., that describe how the eBPF programs are leveraged in enterprise deployments. Finally, the function call graphs are generated from syntactic analysis to assign each function's union of capability labels for the entire eBPF program. An example of capability identification is given in Figure 4.

```
"capabilities": [
    "bpf_get_current_pid_tgid",
    "bpf_get_current_comm"
]
```

Listing (1) Capabilities generated from do_perf_event()

```
"capabilities": [
    "bpf_trace_printk"
]
```

Listing (2) Capabilities generated from print_arg()

```
"capabilities": [
    "bpf_get_current_pid_tgid",
    "bpf_get_current_comm",
    "bpf_trace_printk"
]
```

Listing (3) Capabilities generated from eBPF program consisting of do_perf_event() and print_arg() both

Fig. 4: Representation of generated capabilities labels for an eBPF program.

As shown in Figure 4, let us consider an eBPF program that monitors the performance of various user programs. The program is composed of two functions, namely do_perf_event() and print_arg(). Listing 1 presents the capability labels generated for the function do_perf_event() which is primarily attributed to the use of two helper functions: (a) bpf_get_current_pid_tgid (To obtain the process and thread group id) and (b) bpf_get_current_comm (Obtains command name of current process). Similarly Listing 2 shows the helper function bpf_trace_printk from the function print_arg() which prints formatted output to the */sys/kernel/tracing/trace* file. The overall capability label of the entire eBPF program is depicted in Listing 3, which is a union of the functions mentioned above and represents an overall behavioral profile for the program.

TABLE I: Distribution of extracted capabilities across studied open-source eBPF repositories

Program Capability	Number of functions
pkt_stop_processing_drop	30
pkt_goto_next_module	120
read_sys_info	38
update_pkt	12
read_skb	30
map_read	100
map_update	20

The capabilities of the extracted program are stored in an eBPF registry that provides an online catalog of eBPF functions, extracted capabilities, and meta-data describing their functionality, requirements, and constraints. Table I shows a subset of capabilities extracted from eBPF programs studied in this paper, along with the frequency of their use in our aforementioned open-sourced database. Describing the internal details of the generation of code comments is beyond the scope of this paper and hence omitted for the sake of brevity.

III. Evaluation

The objective of the evaluation is to understand the efficacy, efficiency, and overhead of *BeeGuard* in correctly loading an eBPF program while maintaining policy compliance. We use a server with 2 Intel Xeon Platinum 8358 CPUs and $512GB$ memory for experimental evaluation of *BeeGuard*. We have configured 2 VMs, one for the G-Sent and the other as a client device. Both the VMs have 12 VCPUs and 32GB memory with Linux Kernel 6.1.38 with eBPF support using the bcc [37] framework. The communication latency between the VMs $\approx$ 1ms. During our evaluation, we have used 15 different eBPF programs from open-source repositories [38] as listed in Table II. The programs can be categorized into 3 broad categories based on their functionality.

Kernel Function Tracing: Table IIa contains a list of programs for tracing kernel functions and monitoring system behavior, which are particularly useful for debugging purposes. For example, stacksnoop can capture kernel function's stack traces and help diagnose bottlenecks in system call paths. Additionally, these programs can perform several other diagnostic operations such as measurement of the length of strings called by the *strlen()* function call to monitor string handling (e.g., strlen_count), analyze process creation by adding kprobes on the *clone* system call (e.g., trace_perf), etc. The detailed objectives of these programs are also listed in the table.

Performance Profiling: Table IIb emphasizes the profiling of performance metrics and latency analysis. For example, I/O latency distributions can be measured using biolatpcts, which can help to understand inefficient workloads. Similarly, for disk access pattern, vfslatency helps with providing

TABLE II: List of eBPF programs used for testing.

(a) Kernel Function Tracing

Name of Program	No. of Instructions	Description
stack_buildid	51	Profiles the call stacks of processes using system libraries
stacksnoop	31	Traces a kernel function and prints all kernel stack traces
strlen_count	43	Traces strlen() and print a frequency count of strings
strlen_hist	71	Histogram of system-wide strlen() return values
trace_fields	15	Traces an event and printing custom fields
trace_perf	43	Traces the 'clone' system call using a kprobe

(b) Performance Profiling

Name of Program	No. of Instructions	Description
biolatpcts	69	Calculates IO latency percentile
hello_perf	25	Hello World example that uses BPF_PERF_OUTPUT
hello_perf_ns	36	Hello World example that uses BPF_PERF_OUTPUT with bpf_get_ns_current_pid_tgid()
sync_timing	40	Trace time between syncs
vfslatency	85	VFS read latency distribution

(c) Network Monitoring

Name of Program	No. of Instructions	Description
dddos	62	Using ePBF to detect a potential DDOS attack against a system
nflatency	120	Attaches a kprobe and kretprobe to nf_hook_slow
tcpv4connect	62	Traces TCP IPv4 connect()s
undump	46	Dumps UNIX socket packets

data on file system performance, and sync_timing can be used to perform several other diagnostic operations such as measurement of time taken during synchronization operations to enhance I/O operations.

Network Monitoring: In the last group, as seen in Table IIc, we have eBPF programs used for network monitoring and security analysis. Where dddos can be used to flag potential Denial of Service attacks, nflatency helps improve network understanding by tracking packet processing latency and allows us to pinpoint performance bottlenecks in the network stack. tcpv4connect can be utilized to monitor TCP connection attempts to detect misuse patterns in connectivity.

A. Quantitative Evaluation

To understand the overhead imposed by the framework, we have compared the performance of *BeeGuard* with an equivalent unmodified kernel (UN). We have compared the performance of two versions of *BeeGuard*: (a) *BeeGuard* without the "KV Store" (BG) and (b) *BeeGuard* with the "KV Store" (BG'). All the experiments were repeated 10 times, and the mean and standard deviation (σ) for various time components are reported in Figure 5. We observe that the proposed framework introduces a small overhead ($\approx$ 3ms) during the loading of each eBPF program. This overhead observed is attributed to two sub-actions: (a) Signature calculation from the submitted bytecode and (b) Policy verification.

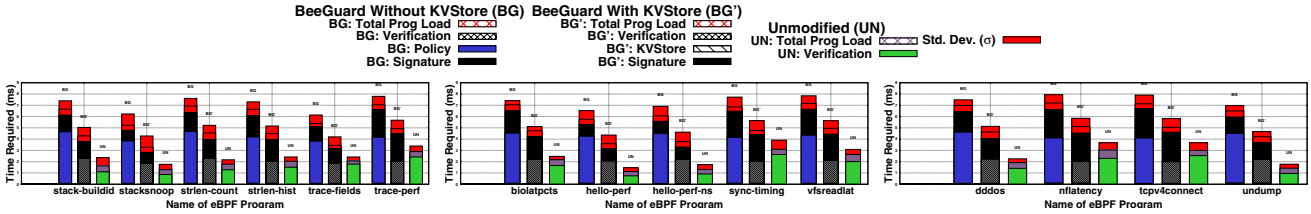
Figure 5 represents the loading time when the policy is calculated in the "KV Store" for the first time. We have observed that while the checksum calculation takes negligible time, the policy verification time at "G-sent" takes significantly longer. As policy compliance for each program can be carried out in parallel, this overhead is not cumulative and is independent of the number of programs executing inside the machine. From the experiments, we found that *BeeGuard* imposes an average overhead of $\approx 4.5 \pm 1$ms for kernel tracing (Figure 5a), and performance profiling (Figure 5b) programs due to added program loading complexity compared to an unmodified kernel. On the other hand, network monitoring programs Figure 5c show an average overhead of $\approx 5 \pm 1$ms. The reason behind the added overhead is that typical network monitoring programs are significantly larger ($\approx 1.5x$) than the other two categories; thus expected to require greater time. We have also observed that the average time to receive a response from the "policy engine" is $\approx 4.8 \pm 1.6$ms for all categories as it is independent of the program categories and size. From the experimental results depicted in Figure 5, we also observe that despite the notably low latency between the policy engine and client devices, the integration of KV Store as a cache can substantially enhance performance by reducing the decision making ($\approx 2ms$ faster). Intuitively, the performance benefit from the KV Store increases if the latency between G-Sent and the client device increases. We have also observed that, due to the signature calculation overhead, the average CPU utilization (see Figure 6) also increased marginally ($< 1\%$) for most of the programs. This increase is most noticeable in network monitoring programs (Figure 6c) and significantly lesser for kernel tracing (Figure 6a) and performance filtering (Figure 6b) programs. This increase in CPU utilization is due to the length of the program. Apart from the enforcement module, we have also measured the resource utilization overhead of the G-Sent micro-services using "docker stat", which reveals (see Figures 7a and 7b) a negligible overhead in terms of CPU and memory footprint.

B. Qualitative Analysis

Although, *BeeGuard* incurs marginal one-time overhead (during program load-time), it offers significant advantages in terms of usability, as discussed below.

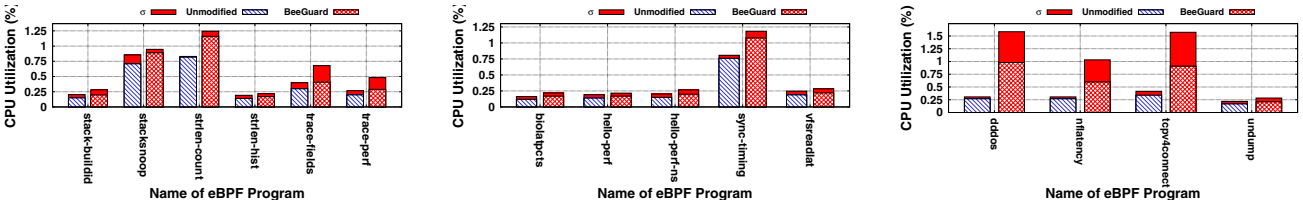
1) Ease of Deployment: All components of *BeeGuard* except the in-kernel enforcer are developed as containers and, thus, are highly scalable and easy to deploy. Moreover, *BeeGuard* can be deployed following the continuous integration/continuous deployment (CI/CD) pipeline.

2) Ease of Policy Modification: Modifying policy in *BeeGuard* is easy due to its use of the policy engine. For example, when a new helper function is added within an eBPF program that is already in the list of predefined policy rules in "p-store", we simply need to update the capability label of the program. A new capability value addition does not require any change in the policy rules, and only a list update in Elastic will serve the purpose. On the other hand, to support the extension of the capability list (i.e., beyond the hook points, tracepoints, and



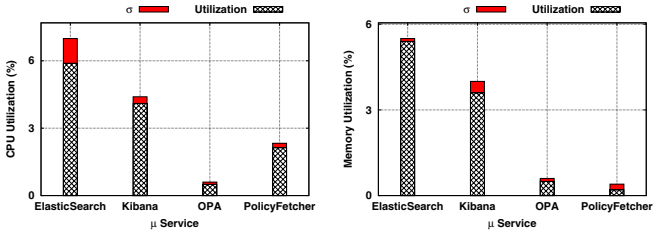
(a) Kernel Function Tracing (b) Performance Profiling (c) Network Monitoring

Fig. 5: Average load time (component-wise) analysis by the three groups of eBPF programs



(a) Kernel Function Tracing (b) Performance Profiling (c) Network Monitoring

Fig. 6: CPU utilization by the three groups of eBPF programs at runtime



(a) CPU utilization **(b) Memory utilization**

Fig. 7: Resource utilization for G-Sent

used helper functions), minor policy rule updation is required ($<$ 10LOC). We have tested these two features by adding a new capability (i.e., eBPF program daisy-chaining using the "pkt_go_to_next_module" helper function) which validates this particular claim.

3) Policy Robustness: Although often used by eBPF programs for monitoring and debugging the system, *tracepoints* can also be used to gain system insights and facilitate attacks. Therefore, the system should leverage the "policy engine" to control the accessibility of the various tracepoints. Additionally, the hookpoint and helper function control also needs to be ensured to avoid open vulnerabilities. In this paper, we have considered three types of policy rules in our "policy engine" to justify the robustness of the framework: (a) block tampered eBPF programs, (b) block side-channeling, and (c) block unrestricted tunneling. We only allow the eBPF bytecodes, which are *untampered* (i.e., the pre-computed signature matches the load time signature). Additionally, we observed that an eBPF program with header parsing capability can increase *side-channel* attack probability [39]. Therefore, any program that uses raw interfaces such as tracepoints is blocked. A rule to block *unrestricted tunneling* [40] has also been developed by restricting header modification, which in turn avoids tunneling-related helper functions. In a nutshell, the last two rules cover the hookpoint and helper function control. The corresponding policy rules are listed in Listing 4. As seen in Line 8 of Listing 4, all capabilities associated with the program are fetched from "p-store" are stored in the list ALL_CAPABILITIES. Two lists (BLKD_TRACEPOINT in Line 12 and BLKD_HELPER in Line 13) are used to store lists of potentially risky tracepoints and helper functions and should be excluded from the allowed list (Lines 15 and 16). We have used two rules to arrive at the policy decision for each eBPF program: (a) setting policy decision as allowed when the program signature is authenticated even though it may have blank capabilities labels; and (b) setting policy decision as allowed when all the capabilities of the program match the ones fetched from "p-store" along with the signature. Furthermore, we have identified all the combinations of these three rules and created customized 8 eBPF programs. For example, we have modified the sources by adding instructions inside the bytecode which should result in a signature mismatch. In a different case, we have added a packet parsing helper function to the strlen_hist to mimic side channeling. We have found that *BeeGuard* only allows legitimate programs to be loaded, and the programs violating policy were blocked.

4) Policy Expressiveness: By modifying suitable policies during the runtime, *BeeGuard* can control the hookpoints or tracepoints and the trusted helper functions. We performed 10 experiments by modifying the policies and observed no deviation from the expected behavior. We have also observed that the policy change takes negligible time ($<$ 10ms) to reflect its effect.

IV. Related Work

A. Existing works using eBPF for profiling

A significant focus of research on eBPF covers the usage of eBPF for profiling or analyzing various aspects of the system

```
### Subroutines
element_belongs_to_list(element, allowed_capabilities) {
    element == allowed_capabilities[_]
}
### Block all programs except explicitly allowed
default allow := false
### Fetch capabilities from p-store
ALL_CAPABILITIES:= sql.send({
    "driver": "sqlite",
    "query": "SELECT capabilities FROM ebpf_registry"
  })
BLKD_TRACEPOINT:=["netif_tx"]
BLKD_HELPER:=["pkt_go_to_next_module", "parse_pkt_headers"]
### List of Allowed capabilities
ALLOWED_CAPABILITIES:=ALL_CAPABILITIES - BLKD_TRACEPOINT
ALLOWED_CAPABILITIES:=ALLOWED_CAPABILITIES - BLKD_HELPER
### Allow programs with blank capability and Matching Signature
allow if{
    count(data._default[input.signature].
     cumulative_capabilities) == 0
}
### Allow programs having allowed capabilities and matching Signature
allow if{
 x =data._default[input.signature].cumulative_capabilities
 every element in x{
  element_belongs_to_list(element,ALLOWED_CAPABILITIES)
 }
}
```

Listing 4: Example of policies used

behavior like performance, networking, etc. Kmon [14] is an early work which proposes an in-kernel monitoring system for microservices which provide various runtime information of containers ranging from latency to performance metrics. eBPF has also been used recently to create application-specific network profiles [18] by using matchers to map processes to their respective applications, thus facilitating identification. From creating packet filtering rules to fit application profiles [10] to profiling the performance of container-based applications [11] or to create custom profiles for containers based on precise system metrics [12], eBPF has been used for system-level profile generation. In *BeeGuard*, we propose the generation of profiles for each eBPF program based on their behavior as inferred from their capability labels.

B. Verifier modifications in the eBPF subsystem

The BPF verifier has been a topic of much interest in academia as seen from recent literature on automated verification to check the soundness of the eBPF verifier for tracking values of its variables [41]. Range analysis research to study the correctness of the range analysis of the verifier for values in its registers [42] and even approaches to identify any illegal behavior in verified code [43] has been explored. However as cited in [44], kernel bugs can also be exploited to bypass the security provided by the verifier thus leading to suggestions to decouple the verification and JIT compilation process from the loader [45]. HyperBee [25] mentions a similar approach to modifying the verification process which requires a malicious program structure database to verify the eBPF programs. Building on this, in *BeeGuard* we exploit the idea of policy enforcement by extending the capacity of the verifier and loader to not just check code for safety but also policy compliance. Unlike, HyperBee, we rely on the helper functions, trace-point/hook-points, etc used in the program to generate the labels.

C. Literature on policy enforcement

Various organizations have always prioritized the need for policy management and with the rapid adaptation of cloud native architecture, security and data privacy have been emerging concerns. KRSI [19] is an architecture which addresses this problem of container runtime security by using eBPF to manage policies across Kubernetes clusters on the fly. Enforcement of policies at runtime for eBPF programs can be improved by modifying the verification process to monitor the runtime cost of programs [20] and laying tighter upper bounds on the runtime. Sauron proposed in [21] explores the idea of rules across multiple pods wherein eBPF is used to monitor cluster control. Similarly, the eBPF-IoT-MUD [22] uses eBPF to enforce manufacturer usage description rules directly in the lower level of the kernel stack. However, unlike [23] which proposes invoking an eBPF program to decide the legitimacy of policies that may not be scalable, in *BeeGuard* we exploit the modified kernel to match and enforce policies. In *BeeGuard* we have experiments to validate that our proposed architecture is independent of the number of programs running.

D. Source control mechanisms for eBPF programs

In contrast to Loadable Kernel Modules (LKMs) which have a signing mechanism that can be verified by the kernel, eBPF subsystem allows any privileged program that passes the verifier checks to load [27], [31]. HyperBee [25] brings forth a signature verifier in it's verifier chain to validate whether the program carries a trusted signature or not. Alternately, in Windows [26] eBPF programs can be compiled as driver images and then consecutively signed using the standard driver signing mechanisms. Since recent literature did not propose any valid source control mechanism, we used the checksum signature verification as proposed in *BeeGuard*.

V. Conclusion and Future Works

This paper proposes a user-friendly and extendable framework for ensuring policy compliance for kernel-level eBPF programs. We observe that the proposed framework is easily extendable (CI/CD compliant) and provides flexibility in terms of usability and future-proofing. This framework has an ingrained capability of extracting the behavioral models from the eBPF sources in terms of capabilities. The current version of the work can be enhanced with an optimized version of local cache access and end-to-end testing to address scalability concerns, which we plan to incorporate in future updates. We also aim to explore the integration of Linux Security Modules (LSM) with the eBPF program to compare the performance benefits with *BeeGuard*. The LSM-based policy enforcement will enhance the current framework's flexibility by conducting security checks at the time of eBPF program loading before entering the verifier.

References

[1] Z. Li, L. Guo, J. Cheng, Q. Chen, B. He, and M. Guo, "The Serverless Computing Survey: A Technical Primer for Design Architecture," *ACM Computing Survey*, vol. 54, no. 10s, pp. 1–34, September 2022.

[2] S. Qi, L. Monis, Z. Zeng, I.-C. Wang, and K. K. Ramakrishnan, "SPRIGHT: High-Performance eBPF-Based Event-Driven, Shared-Memory Processing for Serverless Computing," *IEEE/ACM Transactions on Networking*, vol. 32, no. 3, pp. 2539–2554, 2024.

[3] IBM, "Hyperscaler Cloud Service Providers Top 10," 2021, [accessed 30 September, 2024]. [Online]. Available: https://www.ibm.com/downloads/cas/LYZX6JB5

[4] P. Khan and B. Rajkumar, "A Taxonomy and Survey of Content Delivery Networks," *Grid Computing and Distributed Systems Laboratory, University of Melbourne, Technical Report*, vol. 4, no. 2007, p. 70, 2007.

[5] U. Naseer and T. A. Benson, "Configanator: A Data-driven Approach to Improving CDN Performance," in *USENIX Symposium on Networked Systems Design and Implementation*, 2022, pp. 1135–1158.

[6] A. N. Montanari and L. A. Aguirre, "Observability of Network Systems: A Critical Review of Recent Results," *Journal of Control, Automation and Electrical Systems*, vol. 31, no. 6, pp. 1348–1374, 2020.

[7] M. Usman, S. Ferlin, A. Brunstrom, and J. Taheri, "A Survey on Observability of Distributed Edge & Container-based Microservices," *IEEE Access*, vol. 10, pp. 86 904–86 919, 2022.

[8] eBPF Community, "Extended Berkeley Packet Filters," 2014, [accessed 30 September, 2024]. [Online]. Available: https://ebpf.io/

[9] S. Sundberg, A. Brunstrom, S. Ferlin-Reiter, T. Høiland-Jørgensen, and J. D. Brouer, "Efficient Continuous Latency Monitoring with eBPF," in *Passive and Active Measurement*. Springer Nature, 2023, pp. 191–208.

[10] D. Scholz, D. Raumer, P. Emmerich, A. Kurtz, K. Lesiak, and G. Carle, "Performance Implications of Packet Filtering with Linux eBPF," in *International Teletraffic Congress*, 2018.

[11] C. Cassagnes, L. Trestioreanu, C. Joly, and R. State, "The Rise of eBPF for Non-intrusive Performance Monitoring," in *IEEE/IFIP Network Operations and Management Symposium*, 2020.

[12] J. Levin and T. A. Benson, "ViperProbe: Rethinking Microservice Observability with eBPF," in *IEEE International Conference on Cloud Networking*, 2020.

[13] J. T. Humphries, N. Natu, A. Chaugule, O. Weisse, B. Rhoden, J. Don, L. Rizzo, O. Rombakh, P. Turner, and C. Kozyrakis, "ghOSt: Fast & Flexible User-Space Delegation of Linux Scheduling," in *ACM Symposium on Operating Systems Principles*, 2021, p. 588–604.

[14] T. Weng, W. Yang, G. Yu, P. Chen, J. Cui, and C. Zhang, "Kmon: An In-kernel Transparent Monitoring System for Microservice Systems with eBPF," in *International Workshop on Cloud Intelligence*, 2021.

[15] B. Sharma and D. Nadig, "eBPF-Enhanced Complete Observability Solution for Cloud-native Microservices," in *IEEE International Conference on Communications*, 2024, pp. 1980–1985.

[16] F. J. Bertinatto, D. Arioza, J. C. Nobre, and L. Z. Granville, "Container-Level Auditing in Container Orchestrators with eBPF," in *Advanced Information Networking and Applications*. Springer Nature, 2024, pp. 412–423.

[17] M. Craun, K. Hussain, U. Gautam, Z. Ji, T. Rao, and D. Williams, "Eliminating eBPF Tracing Overhead on Untraced Processes," in *SIGCOMM Workshop on eBPF and Kernel Extensions*. ACM, 2024, pp. 16–22.

[18] L. Wüstrich, M. Schacherbauer, M. Budeus, D. F. von Künßberg, S. Gallenmüller, M.-O. Pahl, and G. Carle, "Network Profiles for Detecting Application-Characteristic Behavior Using Linux eBPF," in *Workshop on eBPF and Kernel Extensions*. ACM, 2023.

[19] S. Gwak, T. P. Doan, and S. Jun, "Container Instrumentation and Enforcement System for Runtime Security of Kubernetes Platform with eBPF," *Intelligent Automation and Soft Computing*, vol. 37, no. 2, pp. 1773–1786, 2023.

[20] R. Sahu and D. Williams, "Enabling BPF Runtime Policies for Better BPF Management," in *Workshop on eBPF and Kernel Extensions*. ACM, 2023, pp. 49–55.

[21] D. Soldani, P. Nahi, H. Bour, S. Jafarizadeh, M. F. Soliman, L. Di Giovanna, F. Monaco, G. Ognibene, and F. Risso, "eBPF: A New Approach to Cloud-Native Observability, Networking and Security for Current (5G) and Future Mobile Networks (6G and Beyond)," *IEEE Access*, vol. 11, pp. 57 174–57 202, 2023.

[22] A. Feraudo, D. A. Popescu, P. Yadav, R. Mortier, and P. Bellavista, "Mitigating IoT Botnet DDoS Attacks through MUD and eBPF based Traffic Filtering," in *International Conference on Distributed Computing and Networking*. ACM, 2024.

[23] J. Jia, M. V. Le, S. Ahmed, D. Williams, and H. Jamjoom, "Practical and Flexible Kernel CFI Enforcement using eBPF," in *Workshop on eBPF and Kernel Extensions*. ACM, 2023, pp. 84–85.

[24] A. Schendel, "Reverse Engineering eBPF Programs: A Deep Dive," https://www.armosec.io/blog/ebpf-reverse-engineering-programs/, 2024, [accessed 30 September, 2024].

[25] Y. Wang, D. Li, and L. Chen, "Seeing the Invisible: Auditing eBPF Programs in Hypervisor with HyperBee," in *Workshop on eBPF and Kernel Extensions*. ACM, 2023.

[26] A. Jowett, "Towards Debuggability and Secure Deployments of eBPF Programs on Windows," https://opensource.microsoft.com/blog/2022/10/25/towards-debuggability-and-secure-deployments-of-ebpf-programs-on-windows/, 2022, [accessed 30 September, 2024].

[27] D. Alden, "Securing BPF Programs Before and After Verification," https://lwn.net/Articles/977394/, 2024, [accessed 30 September, 2024].

[28] Fortinets, "What is DMZ?" https://www.fortinet.com/resources/cyberglossary/what-is-dmz, 2024, [accessed 30 September, 2024].

[29] Elastic, "Elasticsearch: The Official Distributed Search," 2023, [accessed 30 September, 2024]. [Online]. Available: https://www.elastic.co/elasticsearch/

[30] OPA, "Open Policy Agent," 2023, [accessed 30 September, 2024]. [Online]. Available: https://www.openpolicyagent.org/

[31] J. Corbet, "Toward Signed BPF Programs," 2021, [accessed 30 September, 2024]. [Online]. Available: https://lwn.net/Articles/853489/

[32] "BPF Instruction Set Architecture," 2023, [accessed 30 September, 2024]. [Online]. Available: https://www.kernel.org/doc/html/latest/bpf/standardization/instruction-set.html#bpf-instruction-set-architecture-isa

[33] T. H. Le, H. Chen, and M. A. Babar, "Deep Learning for Source Code Modeling and Generation: Models, Applications, and Challenges," *ACM Computing Surveys*, vol. 53, no. 3, pp. 1–38, 2020.

[34] Z. Zhou, H. Yu, and G. Fan, "Adversarial Training and Ensemble Learning for Automatic Code Summarization," *Neural Computing and Applications*, vol. 33, no. 19, pp. 12 571–12 589, 2021.

[35] F. Wang and Y. Shoshitaishvili, "Angr - The Next Generation of Binary Analysis," in *IEEE Cybersecurity Development*, 2017, pp. 8–9.

[36] T. Sharma, M. Kechagia, S. Georgiou, R. Tiwari, I. Vats, H. Moazen, and F. Sarro, "A Survey on Machine Learning Techniques Applied to Source Code," *Journal of Systems and Software*, vol. 209, p. 111934, 2024.

[37] A. Nakryiko, "HOWTO: BCC to libbpf Conversion," 2020, [accessed 30 September, 2024]. [Online]. Available: https://facebookmicrosites.github.io/bpf/blog/2020/02/20/bcc-to-libbpf-howto-guide.html

[38] BCC, "BCC Examples," 2023, [accessed 30 September, 2024]. [Online]. Available: https://github.com/iovisor/bcc/tree/master/examples

[39] MITRE, "Multi-Stage Channels," 2024, [accessed 30 September, 2024]. [Online]. Available: https://attack.mitre.org/techniques/T1104/

[40] Mitre, "Protocol Tunneling," 2024, [accessed 30 September, 2024]. [Online]. Available: https://attack.mitre.org/techniques/T1572/

[41] H. Vishwanathan, M. Shachnai, S. Narayana, and S. Nagarakatte, "Verifying the Verifier: eBPF Range Analysis Verification," in *Computer Aided Verification*. Springer Nature Switzerland, 2023.

[42] S. Bhat and H. Shacham, "Formal Verification of the Linux Kernel eBPF Verifier Range Analysis," 2022. [Online]. Available: https://sanjit-bhat.github.io/assets/pdf/ebpf-verifier-range-analysis22.pdf

[43] H. Sun, Y. Xu, J. Liu, Y. Shen, N. Guan, and Y. Jiang, "Finding Correctness Bugs in eBPF Verifier with Structured and Sanitized Program," in *European Conference on Computer Systems*. ACM, 2024, p. 689–703.

[44] J. Jia, R. Sahu, A. Oswald, D. Williams, M. V. Le, and T. Xu, "Kernel Extension Verification is Untenable," *Workshop on Hot Topics in Operating Systems*, pp. 150–157, 2023.

[45] M. Craun, A. Oswald, and D. Williams, "Enabling eBPF on Embedded Systems Through Decoupled Verification," in *Workshop on eBPF and Kernel Extensions*. ACM, 2023.