Ad Hoc Networks 78 (2018) 24–31

Contents lists available at ScienceDirect

Ad Hoc Networks

journal homepage: www.elsevier.com/locate/adhoc

QoS-based distributed flow management in software defined ultra-dense networks

Tuğçe Bilen[a,*], Kübra Ayvaz[a], Berk Canberk[a,b]

[a] Computer Engineering Department, The Faculty of Computer and Informatics, Istanbul Technical University, Ayazaga, Istanbul, Turkey
[b] Department of Electrical and Computer Engineering, Northeastern University, Boston USA

A R T I C L E I N F O

Keywords:
Ultra-dense networks
Software defined networking
Controller outage
Queuing theory
Mice and Elephant flows
Flow management

A B S T R A C T

Ultra-dense small cell deployment is a promising solution to meet the 1000× throughput improvement desired in next-generation wireless networks. This deployment results in a correspondingly high number of small cells, also increasing the complexity of the architecture. The Software-Defined Networking (SDN) can be used as a solution to ease the management of Ultra-Dense data plane with distributed controllers. However, in a distributed architecture, the load must be balanced among the controllers and an outage in any controller should not damage the management of the network. In order to recover from an outage by considering load distributions, we propose a distributed flow management model in Software-Defined Ultra-Dense Networks based on the queuing theory. In this approach, we model the distributed controllers with different Markovian queuing systems by considering the flow characteristics and outage. Thus, the proposed flow separation module divides the incoming flows of controllers according to the characteristics of mice and elephant during modeling. Correspondingly, incoming mice and elephant flows are modeled by using $M/M/1$ and $M^X/M/1$ systems with additional $M/M/c$ queue until the detection of the outage. The $M/M/c$ queues are used for outage detection thanks to Erlang-C parameter. On the other hand, the $M/M/1$ and $M^X/M/1$ systems are used to load estimations of mice and elephant flows. Therefore, the mice and elephant flows of the outage controllers are transferred to the compensatory controllers by considering the estimated load distributions. Thence, the $M/M/c$ queues of the compensatory controllers are converted to the $M/M^Y/1$ system to satisfy the massive flow traffic with bulk service. With this method, we are able to decrease the waiting times of mice and elephant flows during the outage by 15% and 11% respectively compared to the conventional distributed controller implementation. Moreover, the packet losses of the controllers during the outage are decreased by 32% compared to the conventional implementation.

1. Introduction

Next generation networks will result in an increasing number of mobile devices with high data rate requirements. Some industry estimates, such as the Cisco Visual Networking Index (VNI) report, point towards aggregate mobile data traffic reaching 49 exabytes per month in 2021 [1]. The new infrastructures are developed to satisfy the high data rate demand with low latency. The Ultra-Densification is one of the key infrastructure innovations to enable performance and capacity gain in next-generation networks.

The network densification is a promising architectural solution that can help meet such high data transfer

* Corresponding author.
E-mail addresses: bilent@itu.edu.tr (T. Bilen), kbra.ayvaz@gmail.com (K. Ayvaz), canberk@itu.edu.tr (B. Canberk).

https://doi.org/10.1016/j.adhoc.2018.05.002
1570-8705/© 2018 Elsevier B.V. All rights reserved.

requirements. In such scenarios, the high number of small cells are located within the coverage area of the macrocell. This design results in a low-cost, low-power operation, with a significant increase in the total capacity, coverage, and energy efficiency of the network [2].

The control and management of the excessive number of small cells become hard despite the above-listed benefits of the network densification. One of the key solution to ease the management of this dense network infrastructure is the SDN. Thanks to the SDN, network is separated into control and data planes. Therefore, the management of the data plane is executed by the centralized controller from the control plane based on the OpenFlow protocol without any hardware configuration [3]. However, the centralized controller can be overloaded while meeting the requirements of the Ultra-Dense data plane. As a solution to this problem, the controllers can be used in a distributed manner. However, in this architecture, the load must be balanced among the distributed controllers and an outage in any controller should not prevent the network management.

Thus, through this paper, we aim to compensate the controller outages by considering the flow characteristics and load distributions of all controllers.

1.1. Related work

In current literature, there have been many works to propose different control plane fault management approaches. Gonzalez et al. [4] proposes a mechanism to design a fault-tolerant master-slave SDN controller to balance the consistency and performance. The paper in [5] proposes a prototype SDN controller to tolerate Byzantine faults in the control and data planes. Similarly, Botelho et al. [6] designs a fault-tolerant controller, and materialize it by proposing and formalizing a practical architecture for small to medium-sized networks. The fault-tolerant SDN controller platform processing the control messages transactionally and exactly once is proposed in [7]. Sanchez Vilchez et al. [8] proposes a self-healing based on bayesian networks, but this solution is applied to the centralized SDN infrastructure. The paper in [9] proposes a fast controller failover for multi-domain software-defined networks. The proposed system consists of the controller failure detection and switches reassignment. Also, the timeout delay is used to reduce the detection time of the failures. Obadia et al. [10] proposes two strategies as controller failover mechanisms. Here, the first mechanism uses a greedy algorithm which sends LLDP packets during the failure to search controllers. The second mechanism is based on the pre-partitioning among controllers. The OpenFlow-like pipeline is proposed in [11] for inter-domain failure management. This design uses a detection mechanism based on link probing and reroutes the traffic flows regardless of controller availability. Furthermore, the fault management in software-defined networks is investigated in [12] from different aspects. However, none of these works considers the separation of flows according to the characteristics while compensating the outage. But, this separation decreases the queue waiting times of the flows with different characteristics.

1.2. Contributions

In this paper, we propose a distributed flow management model in software-defined Ultra-Dense networks based on queuing theory to compensate the controller outages. In this approach, distributed controllers are modeled with the Markovian queuing systems by considering the flow characteristics and outage. The main contributions of the proposed method can be listed as follows:

- The proposed model is executed by the Controller Management Unit through the Controller Modeling, Outage Detection, Flow Load Estimation, and Controller Remodeling Modules.
- In Controller Modeling Module, the proposed flow separation module marks the incoming flows according to the characteristics of mice and elephant. Accordingly, incoming mice and elephant flows are modeled by using $M/M/1$ and $M^X/M/1$ systems with additional $M/M/c$ queue until the detection of the outage.
- $M/M/c$ queues are used in the Outage Detection Module to determine the outage controllers thanks to Erlang-C parameter.
- The $M/M/1$ and $M^X/M/1$ systems are used to calculate loads of mice and elephant flows in the Flow Load Estimation Module. In case of the outage, the mice and elephant flows are transferred to other controllers according to estimated load distributions. We also define these as Compensatory Controllers.
- In Controller Remodeling Module, the $M/M/c$ queues of the compensatory controllers are converted to the $M/M^Y/1$ system to satisfy the massive flow traffic with bulk service.

The rest of the paper organized as follows: In Section 2, the proposed network architecture is presented. In Section 3, the flow management model is explained. The proposed mechanism is evaluated in Section 4. Lastly, we conclude the paper in Section 5.

2. Proposed network architecture

In the proposed network architecture, we add a new controller management unit in addition to the traditional control and data planes of the SDN as shown in Fig. 1. The details of the controller management unit and these planes can be explained as follows.

2.1. Control and data planes

In this paper, we define a data plane with a high number of dummy small cells and mobile nodes. These dummy small cells and mobile nodes can be considered as OpenFlow switches and managed by the control plane based on the OpenFlow protocol [13]. The control plane consists of the controllers defined in a distributed manner. The main reason for the distributed controller architecture is to satisfy the needs of the Ultra-Dense data plane. In this paper, we do not perform any data plane optimization. We investigate the controller outage and flow compensation issues from the control plane aspect as explained in the following parts.

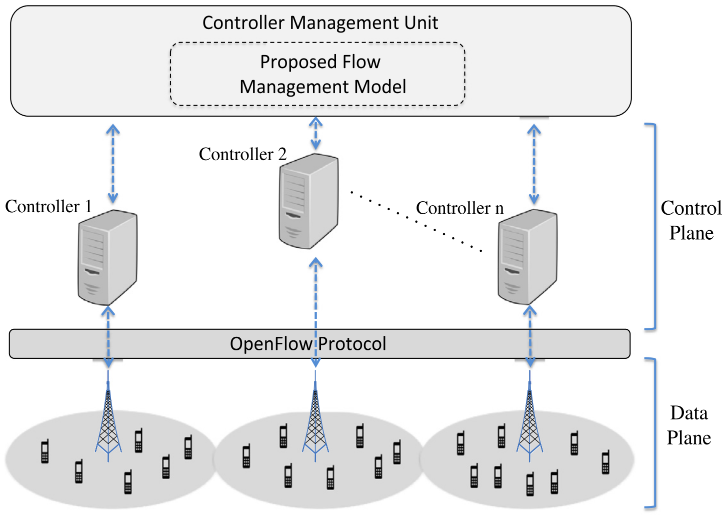

Fig. 1. The proposed network architecture.

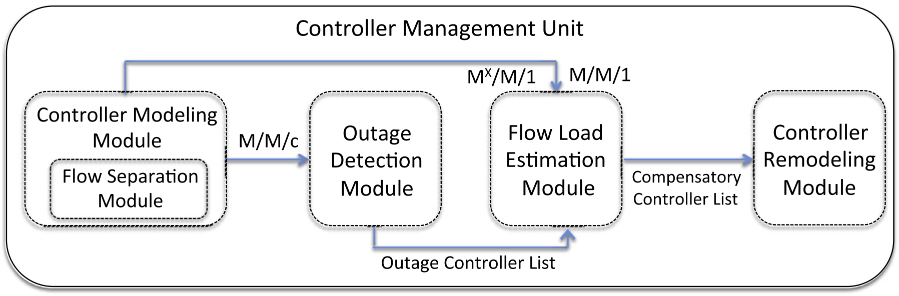
Controller Management Unit
$M^X/M/1$
M/M/1
Controller Modeling Module
Flow Separation Module
M/M/c
Outage Detection Module
Flow Load Estimation Module
Compensatory Controller List
Controller Remodeling Module
Outage Controller List

Fig. 2. Flow management framework.

2.2. Controller management unit

As mentioned above, we define the controllers in a distributed manner to satisfy the requirements of the Ultra-Dense data plane. However, in this distributed architecture, a fault in any controller must be compensated by the other controllers by considering the flow characteristics and load distributions. For this purpose, we define a Controller Management Unit and it consists of the *Controller Modeling, Outage Detection, Flow Load Estimation, and Controller Remodeling Modules*. The Controller Management Unit is responsible for the selection of the compensatory controllers during the outage according to the proposed flow compensation model as detailed in the following section.

3. Flow management model

The proposed flow management model is executed by the Controller Management Unit through the *Controller Modeling, Outage Detection, Flow Load Calculation, and Controller Remodeling Modules* as shown in Fig. 2.

3.1. Controller modeling module

In this module, we model each controller with different Markovian queuing models until the detection of the outage. Also, we define a *Flow Separation Module* in each controller to divide the incoming flows according to the characteristics of mice and elephant.

3.1.1. Flow separation module

The incoming flows cannot reach to the controller in a specific distribution instead they come as random arrivals with high variability. Also, the total arrival stream of flows to each controller is the collection of all flows transferred by the data plane. Therefore, we model the proposed Flow Separation Module according to the *G/G/1* queuing system. Accordingly, the number of incoming flows from one small cell in the data plane to the corresponding controller by time t equal to the $N_f(t)$. The total arrival flow to the controller equals to the $N(t) = \sum_{f=1}^{k} N_f(t)$ and here, k represents the maximum number of sender small cell. Correspondingly, the arrival rate of flows from one small cell in the data plane to the corresponding controller is $\lambda_i \equiv \lim_{t\to\infty} \frac{E[N_f(t)]}{t}$. Equivalently, the mean arrival rate of flows to the corresponding controller $\lambda = \sum_{i=1}^{k} \lambda_i$ equals to $\lim_{t\to\infty} \frac{E[N(t)]}{t}$. After reaching the mean arrival rate λ, we can calculate the squared coefficient of variation of inter-arrival time to the corresponding controller (C_A^2) by using the

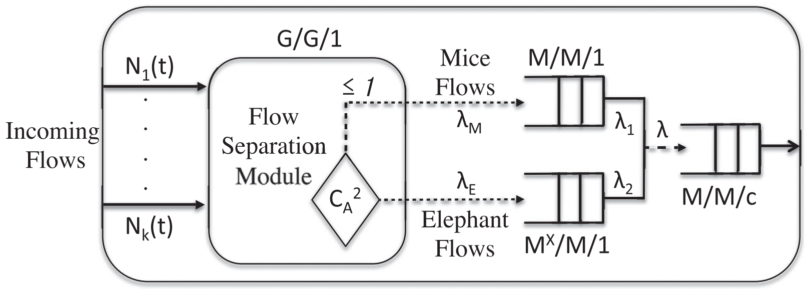

Fig. 3. Controller model until detection of outage.

following equation:

$$C_A^2 = \frac{1}{\lambda}\sum_{i=1}^{k}\lambda_i C_i^2, \quad \forall i \in \mathbb{N} \tag{1}$$

In Eq. (1), C_i^2 is the squared coefficient of variation between the departures from a small cell to the corresponding controller and found as $C_i^2 = \vartheta_i/\lambda_i$. In this equation, the ϑ_i is limiting factor and calculated as $\vartheta_i \equiv \lim_{t\to\infty}\frac{Var[N_f(t)]}{t}$.

The Flow Separation Module calculates the C_A^2 to divide the incoming flows according to the characteristics as mice and elephant. This parameter represents the variation among incoming flows. Therefore, this module marks the incoming flow as an elephant if C_A^2 value is greater than the predefined threshold (it is taken as 1 in this paper). After that, we model the mice flows by using the $M/M/1$ queue. The elephant flows are modeled with $M^X/M/1$ batch model to satisfy the massive incoming traffic characteristic. Because the mice flow is the delay sensitive and carries less packet compared to the elephants. On the other hand, in elephant flows, throughput is more crucial than the delay. Moreover, the $M/M/c$ queue is added next to the queues of mice and elephant flows. In this way, the mice flow with higher priorities is taken queue ahead of elephants as summarized in Algorithm 1. Therefore, each controller is modeled by using the $M/M/1$, $M^X/M/1$, and $M/M/c$ models until the outage is detected as shown in Fig. 3. The details of these queuing models are explained in next sections.

Algorithm 1 Controller Modeling Module.

```
1:  for $i \longleftarrow 1$ to $k$ do
2:      Calculate $C_A^2$ with Eq. 1 for $Flow_i$
3:      if $C_A^2$>1 then
4:          Mark $Flow_i$ as elephant
5:          Assign $Flow_i$ to $M^X/M/1$
6:      else
7:          Mark $Flow_i$ as mice
8:          Assign $Flow_i$ to $M/M/1$
9:      end if
10:     Assign $Flow_i$ to $M/M/c$ from $M/M/1$
11:     if ($M/M/1$ == $\emptyset$) then
12:         Assign $Flow_i$ to $M/M/c$ from $M^X/M/1$
13:     end if
14: end for
```

3.2. Outage detection module

In this paper, the $M/M/c$ system is designed as non-preemptive and the high priority mice flows goes to the head of the queue without any interruption. Also, the mice and elephant flows are transferred to the $M/M/c$ queue according to the Poisson process. The mean arrival rates of the mice and elephant flows to this queue are λ_1 and λ_2, respectively. The total arrival rate is $\lambda = \lambda_1 + \lambda_2$. We also assume that the mean service times of both mice and elephant flows are equal to $\frac{1}{\mu}$. By using these parameters, the steady-state system size probabilities (p_n) of the controllers are found as given in the following equation:

$$p_n = \begin{cases} \frac{\lambda^n}{n!\mu^n}p_0, & 0 \leq n < c \\ \frac{\lambda^n}{c^{n-c}c!\mu^n}p_0, & n \geq c \end{cases}, \quad \forall n \in \mathbb{N} \tag{2}$$

In Eq. (2), $r = \frac{\lambda}{\mu}$ and $\rho = \frac{r}{c}$. Additionally, p_0 is the probability of queue being empty and found by using the following equation:

$$p_0 = \left(\frac{r^c}{c!(1-\rho) + \sum_{n=0}^{c-1}\frac{r^n}{n!}}\right)^{-1}, \quad \forall c, n \in \mathbb{N} \tag{3}$$

We use the queuing delay probabilities of the flows in the corresponding controller to determine the outage. We firstly obtain $W_q(0)$, the probability that flow has zero delays in the queue of the controller. Similarly, $1 - W_q(0)$ represents the probability that flow has a nonzero delay in the queue of the controller. Therefore, $1 - W_q(0)$ shows a probability instead of the direct length of time and this probability is found by using the following equation:

$$W_q(0) = 1 - \frac{r^c p_0}{c!(1-\rho)}, \quad \forall c \in \mathbb{N} \tag{4}$$

Equivalently, the probability that an incoming flow has a nonzero delay in the queue of the controller is found with the following equation:

$$C(c,r) = \frac{\frac{r^c}{c!(1-\rho)}}{\frac{r^c}{c!(1-\rho)} + \sum_{n=0}^{c-1}\frac{r^n}{n!}}, \quad \forall c, n \in \mathbb{N} \tag{5}$$

Eq. (5) can also be defined as *Erlang C* which is the probability that a flow is blocked after waiting a specific length of time in the queue of the controller as explained above. We consider the controller in an outage if this value is greater than the predefined threshold. This threshold is determined according to the Erlang-C table. Consequently, the determined outage controller is transferred to

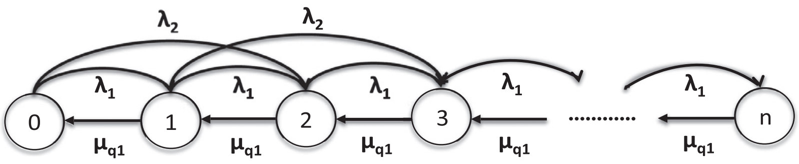

Fig. 4. $M^X/M/1$ queuing model for elephant flows (For 1 or 2 batch sizes).

Algorithm 2 Outage Detection Module.

```
1: for $j \longleftarrow 1$ to $N$ do
2:     Calculate $C(c,r)$ with Eq. 5 for $Controller_j$
3:     if $C(c,r)$ >thrreshold then
4:         Mark $Controller_i$ as outage
5:         Flow Load Estimation ($Controller_i$)
6:     end if
7: end for
```

the Flow Load Estimation Module for the flow compensation as summarized on Algorithm 2.

3.3. Flow load estimation module

In this module, we estimate the mice and elephant flow loads of each controller separately. In this way, we can compensate the outage controller by considering the flow characteristics and load status. First, we analyze the $M/M/1$ queues to determine the mice flow load. Correspondingly, the λ_M and μ are the arrival and service rates of the mice flows. The steady-state probabilities of the $M/M/1$ queue is $p_n = p_0\rho^n,\ \rho = \lambda_M/\mu$. In this equation, p_0 is the probability that no available resource in the controller and found as $p_0 = 1 - \rho$. Accordingly, the mean queue length of the $M/M/1$ queue is found as given in Eq. (6). This estimated queue length also equals to the number of waiting for mice flow load (L_M).

$$L_M = \frac{\lambda_M^2}{\mu(\mu - \lambda_M)} \tag{6}$$

To satisfy the massive traffic characteristics of the incoming elephant flows, we use the $M^X/M/1$ Markovian queuing model as shown in Fig. 4. In this model, λ_n is the arrival rate of elephant flow batches of size n (n represents the packet count in an elephant flow). The total arrival rate (λ_E) equals to the $\sum_{n=1}^{\infty}\lambda_n$. Also, in the $M^X/M/1$ queue, X represents the number of packet per elephant flow batch. With all of these in mind, the number of elephant flows (L_E) in the $M^X/M/1$ queue is found by using the following equation:

$$L_E = \frac{\rho + rE[X^2]}{2(1-\rho)} \tag{7}$$

Also, in this equation, ρ is found as $\rho = \frac{\lambda_E E[X]}{\mu}$. The additional details about the $M^X/M/1$ model can be found in the [14]. Consequently, the mice (L_M) and elephant (L_E) flow loads are estimated for all controllers. After the outage detection, the mice and elephant flows of the outage controller are transferred to compensatory controllers by considering the estimated load distributions. Thus, the

Algorithm 3 Flow Load Estimation Module.

```
1: for $j \longleftarrow 1$ to $N$ do
2:     Calculate $L_{M_j}$ with Eq. 6 for $Controller_j$
3:     Calculate $L_{E_j}$ with Eq. 7 for $Controller_j$
4: end for
5: for $j \longleftarrow 1$ to $N$ do
6:     Find $M/M/1$ queue with min. $L_{M_j}$
7:     Assign outage mice flows to this $M/M/1$
8:     Find $M^X/M/1$ queue with min. $L_{E_j}$
9:     Assign outage elephant flows to this $M^X/M/1$
10:     Controller Remodeling ($Controller_j$)
11: end for
```

mice (or elephant) flows are transferred to the controller with fewer mice (or elephant) flow load as summarized on Algorithm 3. In the following stage, these compensatory controllers are transmitted to the Controller Remodeling Module.

3.4. Controller remodeling module

As detailed above, the queuing flows of the outage controller are transferred to the compensatory controllers according to the flow characteristics and estimated load distributions. However, loads of these compensatory controllers are increased dramatically.

To satisfy this flow load, the queue model of the compensatory controllers should be remodeled as the $M/M^Y/1$ Markovian queuing model as shown in Fig. 5. The controllers with this model give the bulk service and process the flows K at a time. The flows come to $M/M^Y/1$ queue according to the Poisson process. The arrival rates of the mice and elephant flows to the $M/M^Y/1$ queue are λ_1 and λ_2, respectively. The total arrival rate $\lambda = \lambda_1 + \lambda_2$. Additionally, we consider the $M/M^Y/1$ queue according to the partial-batch model. Thus, flows are served K at a time regardless of the number of flow in the queue. The service rate for these flows is the μ.

Accordingly, the waiting time (W_q) of the flows in this queue can be found as $W_q = \frac{r_0}{\lambda(1-r_0)} - \frac{1}{\mu}$. Here, r_0 is any root of the characteristic equation as given in [14].

4. Performance evaluation

4.1. Simulation details

The proposed approach is simulated on the Mininet [15]. The mininet is run on *Ubuntu 14.04* with 8 GB memory, 2,9 GHz Intel Core i7 operating system. The controller

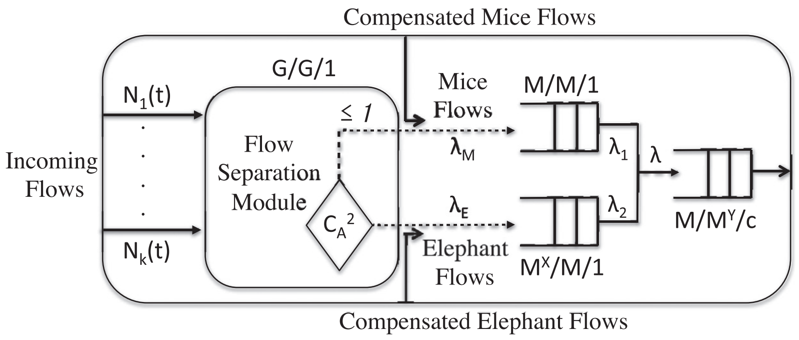

Fig. 5. Controller model after detection of outage (Compensatory Controller).

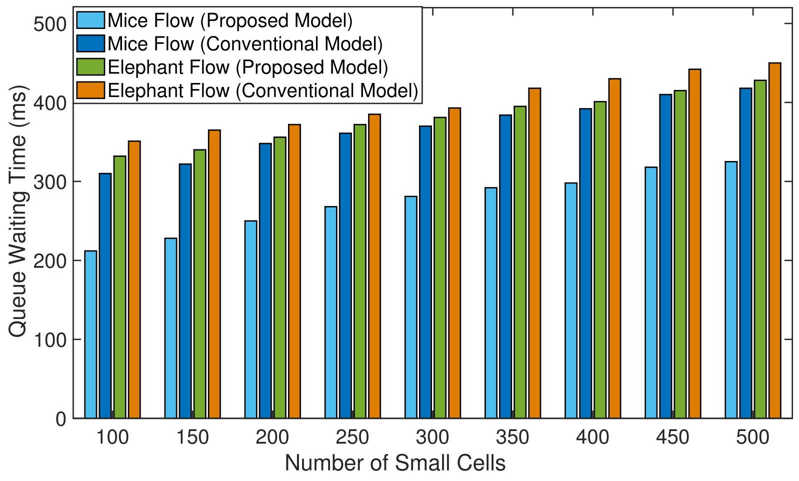

Fig. 6. Mice and elephant flows queue waiting time (During usual operation).

Table 1
Simulation parameters.

Simulation time	600 ms
Receiving Bandwidth	4096 MHz
Noise Rise	6 dB
Maximum/Minimum UE Tx power	21 dBm / −44 dBm
Tx power for macrocells/small cells	30 dBm /20 dBm
Transmission power for macrocells	30 dBm
Transmission power for small cells	20 dBm
Small-cell range	25 m
Macro-cell coverage distance	1 km

is coded by using *POX* [16] in *Python 2.7*[1] language. The *OpenFlow 1.0* protocol[2] is used for communication between Control and Data Plane. Details of the other simulator parameters are given in Table 1.

Additionally, the proposed approach is evaluated by considering the queue waiting times of the mice and elephant flows in controllers for two cases as (i) until the outage (ii) during the outage with increased small cell number. Also, we compare the proposed approach with the conventional distributed controller implementation. The conventional implementation is not modeled by a queuing system according to the flow characteristics.

4.2. Simulation results

In the usual operation of the distributed controllers, we separate incoming flows as mice and elephant to two different queuing system by using Eq. (1). Here, the $M/M/1$ and $M^X/M/1$ queues are used to model the mice and elephant flows, respectively. Then, the $M/M/c$ queue is added next to these systems to prioritize the mice flows against the elephants. This situation decreases the queue waiting time of the mice flows as given in Eq. (4). As shown in Fig. 6, the queue waiting times of the mice flows during the usual operation are 31% less than the conventional distributed controller implementation. Also, the defined $M^X/M/1$ bulk input and $M/M/c$ queues are reduced the waiting times of the elephant flows. Therefore, as shown in Fig. 6, the queue waiting times of the elephant flows are 10% less than the conventional distributed controller implementation during the usual operation. Additionally, the increased small cell number accumulates the incoming flows of the controllers. This also gain the waiting times of the mice and elephant flows.

Moreover, after detection of the outage, the mice and elephant flows are transferred to the selected

[1] Copyright © 2001–2006 Python Network Foundation.
[2] Copyright © 2014, Open Network Foundation.

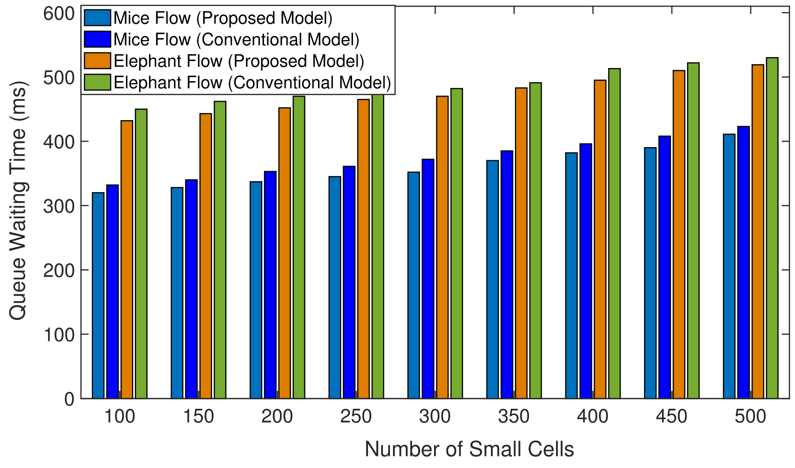

Fig. 7. Mice and elephant flows queue waiting time (during outage).

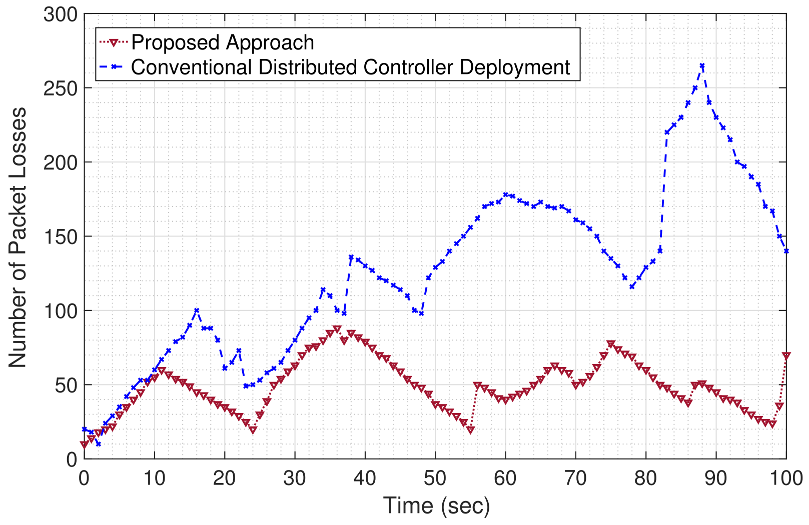

Fig. 8. Packet losses observed during outage.

compensatory controllers. In this selection, the mice flows of the outage controllers are transferred to the $M/M/1$ queue with minimum L_M by using Eq. (6). Equivalently, the elephant flows are transmitted to the $M^X/M/1$ queue with minimum L_E by using Eq. (7). Therefore, as shown in Fig. 7, the waiting times of the mice and elephant flows during the outage are decreased by 15% and 11% compared to the conventional distributed controller implementation.

We also evaluate the observed packet losses of the flows during the outage according to the time. In the proposed approach, we observe the Erlang-C parameters of the queues by using Eq. (5). Correspondingly, the flows are transferred to compensatory controllers before packet losses. Therefore, as shown in Fig. 8, the packet losses observed during the outage are decreased by 32% compared to the conventional implementation.

5. Conclusion

In this paper, we proposed a distributed flow management model for the Ultra-Dense software-defined networks based on Markovian queuing systems. We modeled the distributed controllers based on the incoming flow characteristics and outage status. For this aim, we add a flow separation module to controllers for mice and elephant flow differentiation. Therefore, we can model the mice and elephant flows with $M/M/1$ and $M^X/M/1$ systems with additional $M/M/c$ queue until the detection of an outage. The $M/M/c$ queue is used to detect the outage with Erlang-C parameter. The $M/M/1$ and $M^X/M/1$ systems enable to estimate the mice and elephant flow loads. By using estimated load distributions, the mice and elephant flows in the outage controller are directed to the compensatory controllers. Additionally, to satisfy this massive incoming flow, the $M/M/c$ queues of the compensatory controllers are converted to the $M/M^Y/1$ system. Therefore, the waiting times of the mice and elephant flows during the outage are decreased by 15% and 11% compared to the conventional distributed controller implementation, respectively. Also, the packet losses observed during the outage are decreased by 32% compared to the conventional implementation.

References

[1] White paper: Cisco Visual Networking Index: Global Mobile Data Traffic Forecast Update, 2016–2021 (2017).
[2] J. Hoydis, M. Kobayashi, M. Debbah, Green small-cell networks, IEEE Veh. Technol. Mag. 6 (1) (2011) 37–43, doi:10.1109/MVT.2010.939904.
[3] ONF, Open Networking Foundation, available: https://www.opennetworking.org/ (2014).
[4] A.J. Gonzalez, G. Nencioni, B.E. Helvik, A. Kamisinski, A fault-tolerant and consistent SDN controller, in: IEEE Global Communications Conference (GLOBECOM), 2016, pp. 1–6, doi:10.1109/GLOCOM.2016.7841496.
[5] K. ElDefrawy, T. Kaczmarek, Byzantine fault tolerant software-defined networking controllers, in: IEEE 40th Annual Computer Software and Applications Conference (COMPSAC), 2, 2016, pp. 208–213, doi:10.1109/COMPSAC.2016.76.
[6] F. Botelho, A. Bessani, F.M.V. Ramos, P. Ferreira, On the design of practical fault-tolerant SDN controllers, in: Third European Workshop on Software Defined Networks, 2014, pp. 73–78, doi:10.1109/EWSDN.2014.25.
[7] N. Katta, H. Zhang, M. Freedman, J. Rexford, Ravana: controller fault-tolerance in software-defined networking, in: Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research, in: SOSR '15, ACM, New York, NY, USA, 2015, pp. 4:14:12. doi:10.1145/2774993.2774996.
[8] J. Sanchez Vilchez, I. Grida Ben Yahia, N. Crespi, Poster: Self-Healing Mechanisms for Software-Defined Networks, 2014.
[9] Y.-C. Chan, K. Wang, Y.-H. Hsu, Fast controller failover for multi-domain software-defined networks, in: 2015 European Conference on Networks and Communications (EuCNC), 2015, pp. 370–374, doi:10.1109/EuCNC.2015.7194101.
[10] M. Obadia, M. Bouet, J. Leguay, K. Phemius, L. Iannone, Failover mechanisms for distributed SDN controllers, in: 2014 International Conference and Workshop on the Network of the Future (NOF), Workshop, 2014, pp. 1–6, doi:10.1109/NOF.2014.7119795.
[11] C. Cascone, L. Pollini, D. Sanvito, A. Capone, B. Sanso, SPIDER: fault resilient SDN pipeline with recovery delay guarantees, in: IEEE NetSoft Conference and Workshops (NetSoft), 2016, pp. 296–302.
[12] P.C. d. R. Fonseca, E.S. Mota, A survey on fault management in software-defined networks, IEEE Commun. Surv. Tutor. 19 (4) (2017) 2284–2321, doi:10.1109/COMST.2017.2719862.
[13] Open Networking Foundation, Openflow Switch Specification (2012) Version 1.3.1.
[14] D. Gross, J.F. Shortle, J.M. Thompson, C.M. Harris, Fundamentals of Queueing Theory, 4th, Wiley-Interscience, New York, NY, USA, 2008.
[15] Mininet (http://mininet.org/).
[16] Pox, Pox Openflow Controller, available: http://www.noxrepo.org/pox/about-pox. (2014).

Tuğçe Bilen received her BSc and MSc degrees in Computer Engineering from Istanbul Technical University, Turkey in 2015 and 2017, respectively. She is currently a Phd student in Computer Engineering Program of Istanbul Technical University. She currently serves as a reviewer in IEEE Transactions on Vehicular Technology (TVT), The International Journal of Computer and Telecommunications Networking (COMNET), The Computer Communications (COMCOM). Her research interests include Mobility Management, Content Delivery Networks, Software-Defined Networks and Context Aware Networks.

Kübra Ayvaz received her BS and MS degrees in Computer Engineering from Istanbul Technical University, Turkey, in 2014 and 2017, respectively. She has been working as a Software Engineer at Research and Development Center in Corporate Technology Department, Siemens AS since 2014.

Berk Canberk [S'03, M'11, SM'16] is an Associate Professor at the Department of Computer Engineering in ITU. nce 2016, he is also an Adjunct Associate Professor with the Department of Electrical and Computer Engineering at Northeastern University. He serves as an Editor in IEEE Communications Letter, IEEE Transactions in Vehicular Technology, Elsevier Computer Networks, Elsevier Computer Communications and Wiley International Journal of Communication Systems. He is the recipient of IEEE INFOCOM Best Paper Award (2018), The British Council (UK) Researcher Link Award (2017), IEEE CAMAD Best Paper Award (2016), Royal Academy of Engineering (UK) NEWTON Research Collaboration Award (2015), IEEE INFOCOM Best Poster Paper Award (2015), ITU Successful Faculty Member Award (2015) and Turkish Telecom Collaborative Research Award (2013). His current research areas include Software-Defined Networking (SDN) and Network Function Virtualization (NFV) in 5G Systems, and Next Generation Network Management Systems.