# Distributed Approach to Adaptive SDN Controller Placement Problem

Wei-Li Liu, Li-Hsing Yen
Department of Computer Science,
National Yang Ming Chiao Tung University, Hsinchu, Taiwan.
{s0656539.cs06g, lhyen}@nctu.edu.tw

Tsan-Pin Wang
Department of Computer and Information Science,
National Taichung University, Taichung, Taiwan.
tpwang@mail.ntcu.edu.tw

***Abstract*—In software defined networking (SDN), a controller may manage several SDN switches to be cost-effective while a switch may demand management service from multiple controllers for fault tolerance. The controller placement problem (CPP) is to determine the locations of SDN controllers to minimize the total deployment cost subject to constraints such as controller-switch latency, inter-controller latency, and controller capacity. This problem is challenging especially in interconnected geo-distributed SDN networks. Existing centralized solutions do not well adapt to network dynamics. This paper proposes several distributed mechanisms based on the exact potential game. These mechanisms dynamically adapt to network faults such as link and controller failures. The simulation result shows that these mechanisms need fewer controllers than an existing approach in static networks. When links or controllers may fail, our mechanisms still perform better while only a part of the network nodes is affected. The latter is impossible in non-adaptive approaches.**

## I. Introduction

Software Defined Networking (SDN) decouples control plane from data plane in packet-switching data networks. SDN *controller* on control plane learns of network topology and decides routing for traffic flows. On the other hand, SDN switch on data plane identifies packet flows and applies flow-specific packet forwarding and filtering rules. SDN controller instructs SDN switch to create, update, or delete packet-processing rules and receives notification about unrecognized packets from SDN switch through so-called *southbound* application programming interface (API) such as OpenFlow.

An SDN controller may manage more than one SDN switches subject to its processing capacity. A controller can also manage a remote switch as long as the in-between propagation delay is small enough to meet the reaction-time requirement. We may also connect an SDN switch to more than one controllers for a resilient control plane [1], [2]. When a controller fails, all switches under its management can quickly switch to another controller that meets the capacity constraint and has the lowest worst-case switch-controller latency [3], [4]. In that case, all controllers that manage a common switch should communicate with each other for coordination such as state synchronization. For this reason, the propagation delay between any two such controllers should not exceeded some preset value.

In this paper, we follow the model in [2] and consider a networking architecture where a set of geo-distributed SDN switches are physically interconnected by a wide area network (WAN) (Fig. 1). Each switch generates a certain southbound traffic load and demands some number of controllers to manage it. We have a predetermined set of candidate locations where we can dynamically instantiate or activate SDN controllers to meet the collective demands of all switches. The controller placement problem (CPP) is to determine the set of controller locations so as to minimize overall placement cost while meeting all the mentioned constraints (controller-switch latency, inter-controller latency, and controller capacity).

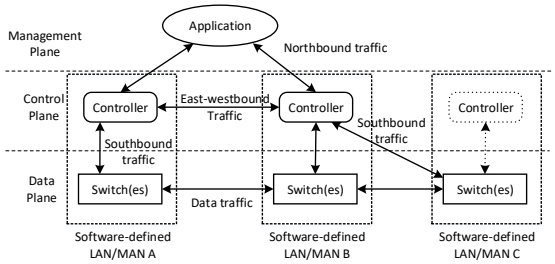



Fig. 1: A network architecture with multiple software-defined networks interconnected by a WAN

The CPP is a variant of capacitated facility location problems. Many existing approaches to CPP [5], [3], [6], [7], [2] aimed to find out the best placement on a static network topology. They did not consider network dynamics such as changes of network topology and traffic conditions. Change of network topology caused by the failure of one or more controllers or links can make an optimal CPP solution suboptimal or even invalid (i.e., the solution no longer meets some constraint). A possible response to the change of network topology is to reconstruct another solution for the new topology. Sallahi and St-Hilaire [8] considered controller capacities and link latency constraints when minimizing the reconstruction cost which accounts for installing controllers, linking switches to the controller, and linking all controllers that manage the same switch. This approach enumerates all possible placements to select the optimal one, which is not time-efficient.

Even if network topology remains unchanged, traffic load may shift over time which makes a fixed configuration suboptimal. ul Huque et al. [9] dealt with dynamic traffic load

by first determining the locations to place controllers and then dynamically adjusting the number of active controllers in each location to adapt to traffic load variations. The approach proposed by Yao et al. [10] mitigates overloaded controllers by handing over some switches from overloaded controllers to other lightly-loaded ones, achieving dynamic load balancing.

Rath et al. [11] proposed a distributed CPP approach where each controller independently determines whether it should switch off (if it is underutilized), offload its workload to other neighboring controllers, or trigger an addition of controllers (if it is overloaded). The authors modeled this approach as a non-zero-sum game. However, the authors did not prove the stability of the game. They neither considered the possibility of multiple controllers managing a common switch.

This paper presents two game-theoretic distributed approaches (and a variant) to CPP. The proposed approaches determine the locations of controllers considering the traffic load generated and the number of controllers demanded by each switch and taking the switch-controller latency, the controller capacity, and the inter-controller latency as constraints. The approaches are autonomous as an agent in each candidate location independently decides whether to instantiate or activate a controller or not. When a link or node fails, the remaining agents can adaptively respond to the change and find a new controller placement in a distributed manner. We prove that the considered game model is an exact potential game (EPG) which guarantees stability regardless of agent's decision making sequence. Simulation results show that one of the proposed approaches demands fewer controllers compared with RCCPP [2]. More importantly, only a part of nodes are affected by link or node failure in all the proposed approaches. This is impossible in non-adaptive approaches such as RCCPP.

The rest of this paper is organized as follows. The next section formulates CPP and reviews related work. Sec. III presents the proposed game designs. Sec. IV shows the experimental results and the final section concludes this paper.

## II. Problem Formulation and Related Work

### A. Problem Formulation

We follow the problem formulation in [2] and consider a network topology $G = (S, E)$, where $S = \{1, 2, \cdots, n\}$ is the set of switch locations and $E$ is the set of network links with weights. The weight associated with a link denotes the latency between the two end locations of the link. We assume that each controller is co-located with some switch. Therefore, $S$ is also the set of locations where a controller can be statically placed or dynamically activated. We refer to the switch placed in location $i$ and the controller (if any) activated or placed in location $i$ as switch $i$ and controller $i$, respectively. For each $i \in S$, $y_i$ indicates whether a controller is indeed placed or activated at location $i$. For each $i, j \in S$, $x_{ij}$ indicates whether controller $i$ (if any) manages switch $j$.

The objective of CPP is to minimize the number of active controllers, i.e.,

$$\min_{\{x_{i,j}\}} \sum_{i \in S} y_i, \tag{1}$$

subject to several constraints. First, both $x_{ij}$ and $y_i$ are binary variables:

$$x_{ij}, y_i \in \{0, 1\}, \forall i, j \in S. \tag{2}$$

Second, no switch can be managed by an inactive controller:

$$y_i \geq x_{ij}, \forall i, j \in S. \tag{3}$$

Third, every switch $j$ needs $t_j$ controllers to manage it, where $t_j \geq 1$ is a positive integer.

$$\sum_{i \in S} x_{ij} = t_j, \forall j \in S. \tag{4}$$

Fourth, the distance between a controller and any switch under its management must not exceed a threshold $SC_{max}$ to meet the controller-switch latency constraint.

$$dist\,(i, j) \cdot x_{ij} \leq SC_{max}, \forall i, j \in S. \tag{5}$$

Fifth, the distance between two controllers $i$ and $j$ that manage a common switch must not exceed a threshold $CC_{max}$ for the synchronization of their states in real time.

$$dist\,(i, j) \cdot x_{ik} \cdot x_{jk} \leq CC_{max}, \forall i, j, k \in S. \tag{6}$$

Finally, every controller $i$ can process at most $c_i$ southbound traffic load. This controller-dependent *capacity constraint* prevents controllers from overloading.

$$\sum_{j \in S} l_j \cdot x_{ij} \leq c_i \cdot y_i, \forall i \in S, \tag{7}$$

where $l_j$ is the southbound traffic load imposed by switch $j$.

### B. Related Work

Killi et al. [12] proposed an approach to CPP which first uses $k$-means algorithm to partition an SDN network. The algorithm then uses cooperative game to form coalitions among switches. Each coalition is a sub-network for controller deployment and the formation is to maximize the value of coalition. Simulation results showed that the worst latency of switch and controller is close to optimal solution.

Heller et al. [5] modeled CPP as $k$-median or $k$-center minimization problem that minimizes the average-case and worst-case controller-switch latency. Yao et al. [6] defined Capacitated Controller Placement Problem (CCPP) and proposed an algorithm to minimize the worst-case propagation delay with the consideration of controller capacity. These two methods obtained placement solutions without considering link or controller failure. Ros and Ruiz [7] defined fault-tolerant CPP, which considers node disconnection due to link failure, and proposed a heuristic algorithm to find a placement that meets the reliability constraint. Killi and Rao [4] proposed a failover mechanism where every switch keeps a backup controller list. When a controller fails, every switch managed by the controller selects a backup controller from its list. Hock et al. [3] proposed Pareto-optimal controller placement (POCO) to find out all potentially good placements. They handled controller failure by deploying multiple controllers for a single switch and considered several objective functions considering

controller failure tolerance, node-to-controller latency, inter-controller latency, and balance of the controller's workloads. However, these works aimed to find out the best placement in a static network topology and did not consider network dynamics.

Meta-heuristic approaches such as Genetic Algorithm (GA), Simulated Annealing (SA) and Particle Swarm Optimization (PSO) can also be used for CPP. Lange et al. [13] proposed a heuristic approach called Pareto Simulated Annealing (PSA). Jalili et al. [14] modeled CPP as a multi-objective optimization problem and used GA to find the Pareto-optimal front of the CPP. This work needs the exact number of controllers as the input. However, CPP generally takes the number of controllers as the output rather than the input of the problem.

Some studies used divide-and-conquer algorithms to solve the CPP [15], [16]. These algorithms split an SDN network into clusters or sub-regions and find the number of needed controllers for each cluster or sub-region. Based on the result, the best switch or location to place the controller in each cluster or sub-region is then determined.

Tanha et al. [2] proposed a heuristic algorithm to minimize the number of controllers according to the controller capacity, switch-controller latency, and inter-controller latency. It takes switch traffic load and controller resilience (failure) as two constraints. Based on the clique concept, each switch and its controllers form a clique such that each link connecting any pair of them must meet the latency constraint. The proposed approach finds out all maximal cliques and used a greedy approach to assign switches to active controllers. Bari et al. [17] studied a dynamic version of CPP where the flow setup changes over time. They proposed a heuristic algorithm to find a placement that minimizes the aggregate cost of status gathering, flow setup, synchronization, and switch reassignment.

## III. Proposed Distributed Approaches

### A. Basic CPP Game

Let $P = \{p_1, p_2, ..., p_n\}$ be the set of all agents in the game, where $p_i$ is for switch location $i \in S$. The *strategy* (i.e., decision choices) of each agent $p_i$ is to determine whether controller $i$ (if any) should manage each switch $j$ in the network. The strategy is essentially a decision vector $\mathbf{x}_i = (x_{i1}, x_{i2}, \cdots, x_{in})$, where $x_{ij} = 1$ if controller $i$ manages switch $j$ and $x_{ij} = 0$ otherwise. We have $\mathbf{x}_i = \mathbf{0}$ if and only if $p_i$ decides not to deploy or activate controller $i$. Let $F_{ij} = \{0, 1\}$ be the range of $x_{ij}$. Strategy set $\Omega_i = F_{i1} \times F_{i2} \times \cdots \times F_{in}$ is the set of $p_i$'s feasible strategies. Let $u_i(\cdot)$ be $p_i$'s utility. The CPP game is defined as a three-tuple $\Gamma = (P, \{\Omega_i\}_{i=1}^n, \{u_i(\cdot)\}_{i=1}^n)$.

The strategy space of the game $\Omega = \Omega_1 \times \Omega_2 \times \cdots \times \Omega_n$ is the Cartesian product of all agent's strategy sets. A strategy profile $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \cdots, \mathbf{x}_n) \in \Omega$ is a possible game state and also a potential solution to CPP.

The objective shown in (1) is to find a game state $\mathbf{x} \in \Omega$ that minimizes the number of active controllers. As a non-cooperative game, where agents do not cooperate with each other, we decompose the global objective into local objectives, one for each agent. Let $u_i(\mathbf{x})$ be $p_i$'s utility in a game state $\mathbf{x}$. The local objective of each agent $p_i \in P$ is to maximize its utility, i.e.,

$$\max_{\mathbf{x}_i \in \Omega_i} u_i\left(\mathbf{x}_i, \mathbf{x}_{-i}\right), \tag{8}$$

where $\mathbf{x}_{-i}$ denotes the set of all agent's strategies other than $p_i$'s.

Our design centers on the gain of utility that a controller may have when it manages a particular switch. More explicitly, $p_i$ will be granted a utility gain of $\alpha$ (a positive constant) from managing switch $j$ (i.e., $x_{ij} = 1$) if the management is *really needed* and also *valid*.

In a game state $\mathbf{x}$, a controller management on switch $j$ is really needed if and only if $v_j(\mathbf{x}) \leq t_j$, where $v_j(\mathbf{x})$ counts the number of *qualified* controllers that decide to manage switch $j$. Controller $i$ is qualified to manage switch $j$ if the distance between them does not exceed $SC_{max}$. Formally,

$$v_j\left(\mathbf{x}\right) = \sum_{i=1}^{n} x_{ij} Z_{ij}, \tag{9}$$

where

$$Z_{ij} = \begin{cases} 1, & \text{if } \mathit{dist}\,(i,j) \leq SC_{max} \\ 0, & \text{otherwise.} \end{cases} \tag{10}$$

Each valid management on switch $j$ receives a gain $g_j\left(\mathbf{x}\right)$ defined as

$$g_j\left(\mathbf{x}\right) = \begin{cases} \alpha & \text{if } v_j(\mathbf{x}) \leq t_j, \\ 0 & \text{otherwise,} \end{cases} \tag{11}$$

where $\alpha > 0$ is a constant. Eq. (11) implies that no controller can benefit from its management on $j$ if excessive controllers decide to manage $j$ at the same time, which motivates some controller to retract the management decision.

Controller $i$'s management on switch $j$ is valid if $i$ is qualified to manage $j$ (i.e., $Z_{ij} = 1$) and the management also meets the capacity and the inter-controller latency constraints. For capacity constraint, function $k_i\left(\mathbf{x}_i\right)$ indicates whether the aggregated load of switches managed by controller $i$ is still within its capacity $c_i$:

$$k_i\left(\mathbf{x}_i\right) = \begin{cases} 1, & \text{if } \sum\limits_{j=1}^{n} x_{ij} l_j \leq c_i, \\ 0, & \text{otherwise.} \end{cases} \tag{12}$$

For the inter-controller latency constraint, let $Y_{ik}$ indicate whether the distance between controllers $i$ and $k$ exceeds the threshold $CC_{max}$ that corresponds to the inter-controller latency constraint:

$$Y_{ik} = \begin{cases} 1 & \text{if } \mathit{dist}\,(i,k) > CC_{max}, \\ 0 & \text{otherwise.} \end{cases} \tag{13}$$

Function $Con_{ij}(\mathbf{x})$ indicates whether there exists any other controller $k$ that also manages switch $j$ but with a distance to controller $i$ larger than $CC_{max}$. It is 1 if there is no such controller and 0 otherwise. Formally,

$$Con_{ij}(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum\limits_{k \neq i} x_{kj} Z_{kj} Y_{ik} = 0, \\ 0 & \text{otherwise.} \end{cases} \tag{14}$$

Now the utility function $u_i(\mathbf{x})$ for every agent $p_i \in P$ can be defined as

$$u_i(\mathbf{x}) = \sum_{j=1}^{n} x_{ij} \left( k_i(\mathbf{x}_i) Z_{ij} g_j(\mathbf{x}) Con_{ij}(\mathbf{x}) - \beta \right), \tag{15}$$

where $0 < \beta < \alpha$ is another constant that denotes the cost to manage a switch. Because $0 < \beta < \alpha$, $p_i$'s management on switch $j$ is profitable only if the management is valid and really needed (i.e., $k_i(\mathbf{x}_i) Z_{ij} g_j(\mathbf{x}) Con_{ij}(\mathbf{x}) = \alpha$).

We can derive the following non-deterministic protocol from the defined game model. We say that agents $i$ and $k$ are neighbors if $Z_{ij} Z_{kj} = 1$ for some $j \in S$. Each agent initializes its strategy with an arbitrary value and inform all its neighbors of the strategy. Afterwards, every agent $p_i \in P$ can independently choose $\mathbf{x}_i \in \Omega_i$ to maximize $u_i(\mathbf{x}_i, \mathbf{x}_{-i})$, but only one agent can do so at a time. All agents are myopic in the sense that they only choose the best strategies that maximize their utilities with respect to the current game state without considering the impact of their choices on future play in the game. The myopic best response function defines the best-response set of strategies

$$\mathcal{R}_i(\mathbf{x}_{-i}) = \{ \mathbf{x}_i \in \Omega_i \mid \forall \mathbf{x}'_i \in \Omega_i, u_i(\mathbf{x}_i, \mathbf{x}_{-i}) \geq u_i(\mathbf{x}'_i, \mathbf{x}_{-i}) \} \tag{16}$$

for every agent $p_i \in P$.

Whenever an agent $p_i$ detects that $\mathbf{x}_i \notin \mathcal{R}_i(\mathbf{x}_{-i})$, it changes its strategy to some strategy in $\mathcal{R}_i(\mathbf{x}_{-i})$ and informs all its neighbors of the change. That action may trigger other agent's reactions. When network condition changes due to events such as link or controller failure, agents may detect the failure locally or receive strategies updates from other agents. In either case, agents may change their strategies locally as their responses and notify all their neighbors of such changes.

### B. Stability of The Basic CPP Game

An agent's strategy change causes a state transition of the game. A sequence of game state transitions each caused by an agent's best-response action can be finite or infinite. A sequence is finite if and only if it ends at a game state where no agent can further improve its utility by changing its strategy. Such a game state is a *Nash equilibrium* (NE). Formally, a strategy profile $\mathbf{x}^* = (\mathbf{x}_1^*, \mathbf{x}_2^*, \cdots, \mathbf{x}_n^*) \in \Omega$ is an NE if $\mathbf{x}_i^* \in \mathcal{R}_i(\mathbf{x}_{-i})$ for all $p_i \in P$.

An NE corresponds to a *stable* solution of the game. However, as agents autonomously change strategies, the resulting game state transitions are non-deterministic. We must ensure that *every* possible sequence of game state transitions is finite, i.e., eventually leads to an NE despite the non-determinism of best-response dynamics.

In the following, we prove that the CPP game $\Gamma$ always ends up with an NE. First, (12) implies that each agent $p_i$ will only manage a subset of $S$ to maximize its utility (15). Each controller with a valid management on the same switch $j$ will receive the same amount of payoff $g_j(\mathbf{x})$, which is a function of the number of valid managements on $j$. Also, the utility of $p_i$ is the sum of the payoffs it can receive from all switches under its management. These two characteristics make $\Gamma$ a congestion game. It is known that every congestion game is an exact potential game (EPG), which admits an exact potential function $\phi(\mathbf{x})$ such that $\forall p_i \in P$, $\forall \mathbf{x}_i, \mathbf{x}_i^* \in \Omega_i$,

$$u_i(\mathbf{x}_i^*, \mathbf{x}_{-i}) - u_i(\mathbf{x}_i, \mathbf{x}_{-i}) = \phi(\mathbf{x}_i^*, \mathbf{x}_{-i}) - \phi(\mathbf{x}_i, \mathbf{x}_{-i}). \tag{17}$$

The exact potential function of $\Gamma$ is

$$\phi(\mathbf{x}) = \sum_{j=1}^{n} \sum_{k=0}^{w_j(\mathbf{x})} \delta(j,k) - \beta \sum_{i=1}^{n} \sum_{j=1}^{n} x_{ij}, \tag{18}$$

where $w_j(\mathbf{x}) = \sum_{i=1}^{n} x_{ij}$ and

$$\delta(j,k) = \begin{cases} \alpha & \text{if } 0 < k \leq t_j \\ 0 & \text{otherwise.} \end{cases} \tag{19}$$

By [18], $\Gamma$ possesses at least one NE and the best-response dynamics will lead the game into an NE.

### C. Prioritized CPP (PCPP) and Quick PCPP Games

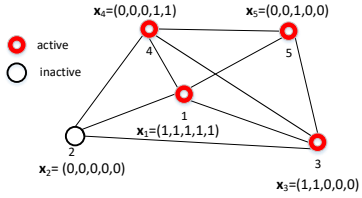



Fig. 2: An NE with four active controllers in the basic CPP game. Each edge $(i,j)$ in the network topology indicates that a controller placed in location $i$ (resp. $j$) is qualified to manage the switch in location $j$ (resp. $i$).

Although the basic CPP game guarantees stability, it may not yield the optimal result. Fig. 2 shows an example where four controllers are activated. Here we assume that the capacity and latency constraints are all met and $t_j = 2$ for all $j \in S$. The game state is already an NE and also a solution to CPP. However, provided that all the constraints are still met, there exists another NE with only two active controllers (Fig. 3).

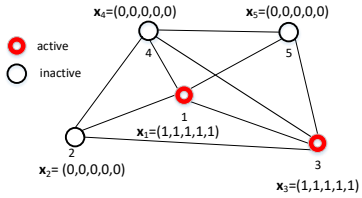



Fig. 3: Another NE with two active controllers in the basic CPP game.

One way to induce a sequence of game state transitions from the NE shown in Fig. 2 to that shown in Fig. 3 is to redefine the game utility function. We refer to the new game as prioritized CPP (PCPP) game. The basic idea of PCPP game is try to utilize controllers that already manage more switches than their peers. To this end, we define

$$S_i = \left\{ j \;\middle|\; \sum_{k=1}^{n} Z_{jk} x_{jk} \geq \sum_{k=1}^{n} Z_{ik} x_{ik} \right\} \tag{20}$$

to be the set of locations where controllers manage no fewer switches than controller $i$. We then redefine (9) so that when $p_i$ counts the number of qualified controllers that decide to manage switch $j$, it only counts those in $S_i$:

$$v_{ij}(\mathbf{x}) = \sum_{k \in S_i} x_{kj} Z_{kj}. \tag{21}$$

Similarly, we also redefine $Con_{ij}(\mathbf{x})$ for the new game.

$$Con_{ij}(\mathbf{x}) = \begin{cases} 1 & \text{if } \sum_{k \in S_i} x_{kj} Z_{kj} Y_{ik} = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{22}$$

In the basic CPP game, a controller management on any switch cannot be preempted by any other controller. In the PCPP game, controller $i$ can still decide to manage switch $j$ as long as the number of qualified controllers in $S_i$ that decide to manage $j$ does not exceed $t_j - 1$. For example, though controllers 1 and 5 already manage switch 3 in Fig. 2, controller 3 can still decide to manage switch 3 because it sees only one controller (i.e., controller 1) that already manages the same switch (as $v_{33}(\mathbf{x}) = 1$). Afterwards, the best response of controller 5 is to retract its management on switch 3, resulting in only three active controllers. Further state transitions can lead the game into the result shown in Fig. 3.

Each agent in the PCPP game need $O(2^n)$ time to find its best response. We thus consider a variant of the PCPP game: Quick PCPP. When making a strategy, each agent $p_i$ first tests if managing each individual switch $j$ is worthy. The answer is positive if $x_{ij} Z_{ij} g_j(\mathbf{x}) Con_{ij}(\mathbf{x}) = \alpha$ (after setting $x_{ij} = 1$). Let $\Phi_i$ be the set of switches for which the management of controller $i$ is worthy. If the aggregate workload in $\Phi_i$ does not exceed $c_i$, then $\Phi_i$ will be the switch set that controller $i$ will manage. Otherwise, $p_i$ repeatedly removes a switch with the most workload from $\Phi_i$ until the aggregate workload no longer exceeds $c_i$. The final $\Phi_i$ will then be $p_i$'s strategy. In this way, the strategy making time reduces to $O(n^2)$.

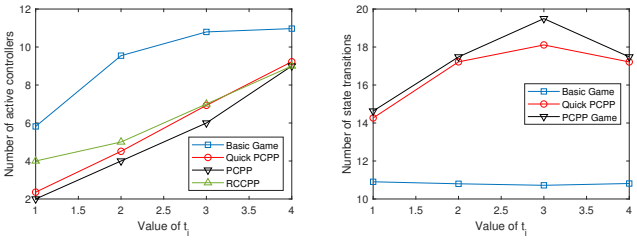
## IV. Numerical Results

### A. Simulation Setup

We used the Sprint topology in [19] as our WAN topology and followed the setting in [2]. Since every node in the topology is a city in the U.S., we calculated the physical distance between any two cities based on their longitude and latitude coordinates. Let $d_{\max}$ be the largest distance between any two cities. $SC_{max}$ was set to $0.4 \times d_{\max}$ whereas $CC_{max}$ was $0.8 \times d_{\max}$. Other default settings were $c_i = 2000$ k req/s and $t_i = 2$ for all $i \in S$. The values of $l_i$'s were exponentially distributed with mean 200 k req/s. Since game play sequences are non-deterministic, all results are averaged over 200 trials.

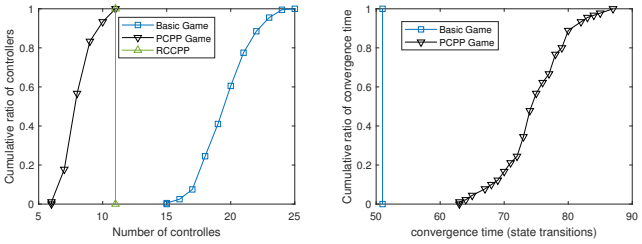
### B. Static Topology

We changed the mean value of $t_i$ in the first experiments. We modified RCCPP [2] for performance comparison with the proposed approaches. Fig. 4a shows the number of active controllers found by each approach. In general, more controllers were needed as the mean of $t_i$ increased. Among all




(a) Number of controllers (b) Convergence time

Fig. 4: Results with different settings of $t_i$




(a) Number of controllers (b) Convergence time

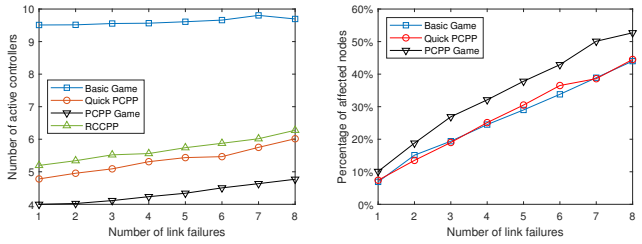
Fig. 5: Results in the DFN topology

approaches, the PCPP game yielded the fewest controllers, followed by the Quick PCPP game and then RCCPP. The Basic CPP game needs the most controllers, which is expected as it is not designed to minimize the number of active controllers. Fig. 4b shows the convergence time of our game-based approaches in terms of the average length of game play sequences. Here the Basic CPP game outperforms the others. The Quick PCPP game performs slightly better than the PCPP game. The "no free lunch" theorem applies here considering the result shown in Fig. 4a.

For large-scale network, we used the DFN topology in [19]. We changed $c_i$ to 10000 k req/s and kept all the other settings. Fig. 5a shows the cumulative ratios of controllers in the result. RCCPP always needs 11 controllers. By contrast, the PCPP game needs fewer while the basic CPP game needs more. Fig. 5b compares these games in terms of convergence time. We can see that the superiority of the PCPP game over the basic CPP game comes at the cost of convergence time.

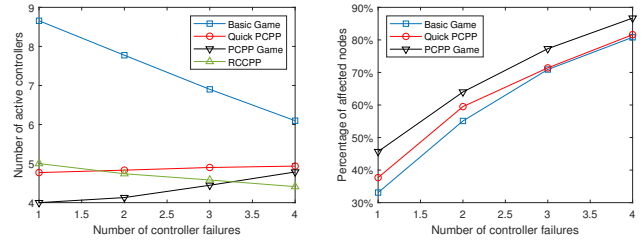
### C. Link and Controller Failures

To simulate a link failure, we first generated a CPP solution for the given static network. We then randomly chose a link and manually made the link latency larger than $SC_{max}$. We increased the number of link failures and observed how the number of active controllers changes. Fig. 6a shows the result. The Basic CPP game is more resilient to link failures than the others because it normally activates excessive controllers. All the other schemes exhibit the same increasing trend and their rankings remain unchanged. Since RCCPP does not react to link failure, we reran RCCPP every time we introduced link failures into the network. Therefore, all nodes were affected by link failures in RCCPP. By contrast, only a part of agents




(a) Number of controllers (b) Percentage of affected nodes

Fig. 6: Impact of link failure




(a) Number of controllers (b) Percentage of affected nodes

Fig. 7: Impact of controller failure

changed their strategies when link failures occurred. Fig. 6b shows the ratios of nodes (i.e., agents) affected by link failures (i.e., ever changing their strategies due to link failures) in all game-based mechanisms. The Basic CPP and the Quick PCPP games both have lower ratios than the PCPP game. Nevertheless, no more than $50\%$ nodes were affected in all game-based mechanisms even with eight failed links.

We simulated a controller failure by randomly removing an active controller (together with the co-located switch) from a CPP solution. Fig. 7a shows the resulting numbers of active controllers with an increasing number of controller failures. The number with the Basic CPP game decreases linearly with the number of controller failures. This can be explained as the survival controllers suffice to take over the switches managed by failed controllers. By contrast, the PCPP and the Quick PCPP games recruited additional controllers because the survival controllers are almost fully utilized. RCCPP needs fewer controllers simply because we regenerated a new solution for the same network with fewer switches.

Fig. 7b shows the ratios of nodes affected by controller failures. Compared with Fig. 6b, we can see that the impact of controller failures is severer than that of link failures. When four controllers failed, the ratio can be near $90\%$. Similar to the case of link failures, the Basic CPP and the Quick PCPP games both have lower ratios than the PCPP game.

## V. Conclusions

We have proposed three game-theoretic approaches to CPP: basic CPP, PCPP, and Quick PCPP games. The basic CPP game has the shortest convergence time but activates the most controllers. On the other hand, the PCPP game activates the fewest controllers but also induces the longest convergence time. The Quick PCPP game is a trade-off between these two extremes. In static networks, the PCPP and the Quick PCPP games can activate fewer controllers than RCCPP [2]. When link or controller failure may occur, the PCPP game outperforms all the other approaches in terms of the number of active controllers. More importantly, all game-theoretic approaches exhibit a strong point that link or node failures only affect a part of nodes in the network.

## References


[1] R. Ahmed and R. Boutaba, "Design considerations for managing wide area software defined networks," *IEEE Commun. Mag.*, vol. 52, no. 7, p. 116–123, Jul. 2014.

[2] M. Tanha, D. Sajjadi, R. Ruby, and J. Pan, "Capacity-aware and delay-guaranteed resilient controller placement for software-defined WANs," *IEEE Trans. Netw. Service Manag.*, vol. 15, no. 3, pp. 991–1005, Sep. 2018.

[3] D. Hock, M. Hartmann, S. Gebert, M. Jarschel, T. Zinner, and P. Tran-Gia, "Pareto-optimal resilient controller placement in SDN-based core networks," in *Proc. 25th Int'l Teletraffic Congress (ITC)*, Sep. 2013, pp. 1–9.

[4] B. P. R. Killi and S. V. Rao, "Optimal model for failure foresight capacitated controller placement in software-defined networks," *IEEE Commun. Lett.*, vol. 20, no. 6, pp. 1108–1111, Jun. 2016.

[5] B. Heller, R. Sherwood, and N. McKeown, "The controller placement problem," in *Proc. 1st Workshop on Hot Topics in Software Defined Networks*, 2012, pp. 7–12.

[6] G. Yao, J. Bi, Y. Li, and L. Guo, "On the capacitated controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 18, no. 8, pp. 1339–1342, Aug. 2014.

[7] F. J. Ros and P. M. Ruiz, "On reliable controller placements in software-defined networks," *Comput. Commun.*, vol. 77, pp. 41–51, 2016.

[8] A. Sallahi and M. St-Hilaire, "Expansion model for the controller placement problem in software defined networks," *IEEE Commun. Lett.*, vol. 21, no. 2, pp. 274–277, Feb. 2017.

[9] M. T. I. ul Huque, G. Jourjon, and V. Gramoli, "Revisiting the controller placement problem," in *Proc. IEEE 40th Conf. on Local Comput. Netw.*, Oct. 2015, pp. 450–453.

[10] L. Yao, P. Hong, W. Zhang, J. Li, and D. Ni, "Controller placement and flow based dynamic management problem towards SDN," in *IEEE Int'l Conf. on Commun. Workshop*, Jun. 2015, pp. 363–368.

[11] H. K. Rath, V. Revoori, S. M. Nadaf, and A. Simha, "Optimal controller placement in software defined networks (SDN) using a non-zero-sum game," in *Proc. IEEE Int'l Symp. on a World of Wireless, Mobile and Multimedia Netw.*, Jun. 2014, pp. 1–6.

[12] B. P. R. Killi, E. A. Reddy, and S. V. Rao, "Cooperative game theory based network partitioning for controller placement in SDN," in *Proc. 10th Int'l Conf. on Commun. Syst. Netw.*, Jan. 2018, pp. 105–112.

[13] S. Lange, S. Gebert, T. Zinner, P. Tran-Gia, D. Hock, M. Jarschel, and M. Hoffmann, "Heuristic approaches to the controller placement problem in large scale SDN networks," *IEEE Trans. Netw. Service Manag.*, vol. 12, no. 1, pp. 4–17, Mar. 2015.

[14] A. Jalili, V. Ahmadi, M. Keshtgari, and M. Kazemi, "Controller placement in software-defined WAN using multi objective genetic algorithm," in *2nd Int'l Conf. on Knowledge-Based Engineering and Innovation*, Nov. 2015, pp. 656–662.

[15] J. Liao, H. Sun, J. Wang, Q. Qi, K. Li, and T. Li, "Density cluster based approach for controller placement problem in large-scale software defined networkings," *Comput. Netw.*, vol. 112, pp. 24–35, 2017.

[16] M. T. I. ul Huque, W. Si, G. Jourjon, and V. Gramoli, "Large-scale dynamic controller placement," *IEEE Trans. Netw. Service Manag.*, vol. 14, no. 1, pp. 63–76, Mar. 2017.

[17] M. F. Bari, A. R. Roy, S. R. Chowdhury, Q. Zhang, M. F. Zhani, R. Ahmed, and R. Boutaba, "Dynamic controller provisioning in software defined networks," in *Proc. 9th Int'l Conf. on Netw. and Service Manag.*, Oct. 2013, pp. 18–25.

[18] D. Monderer and L. S. Shapley, "Potential games," *Games and Economic Behavior*, vol. 14, no. 1, pp. 124 – 143, 1996.

[19] S. Knight, H. X. Nguyen, N. Falkner, R. Bowden, and M. Roughan, "The Internet topology Zoo," *IEEE J. Sel. Areas Commun.*, vol. 29, no. 9, pp. 1765–1775, Oct. 2011.