sensors

MDPI

Article

Data-Driven Packet Loss Estimation for Node Healthy Sensing in Decentralized Cluster

Hangyu Fan, Huandong Wang and Yong Li *

Department of Electronic Engineering, Tsinghua University, Beijing 100084, China;
fhy14@mails.tsinghua.edu.cn (H.F.); whd14@mails.tsinghua.edu.cn (H.W.)
* Correspondence: liyong07@tsinghua.edu.cn

Received: 14 November 2017; Accepted: 12 January 2018; Published: 23 January 2018

Abstract: Decentralized clustering of modern information technology is widely adopted in various fields these years. One of the main reason is the features of high availability and the failure-tolerance which can prevent the entire system form broking down by a failure of a single point. Recently, toolkits such as Akka are used by the public commonly to easily build such kind of cluster. However, clusters of such kind that use Gossip as their membership managing protocol and use link failure detecting mechanism to detect link failures cannot deal with the scenario that a node stochastically drops packets and corrupts the member status of the cluster. In this paper, we formulate the problem to be evaluating the link quality and finding a max clique (NP-Complete) in the connectivity graph. We then proposed an algorithm that consists of two models driven by data from application layer to respectively solving these two problems. Through simulations with statistical data and a real-world product, we demonstrate that our algorithm has a good performance.

Keywords: failure detection; distributed system; gossip protocol; stochastic packet loss

1. Introduction

Clustering technologies leverage a set of connected computers to work as a single system [1], which improves performance, fault-tolerance and scalability of the system. It is extremely important in areas such as sensor networking, clouding computing, centralized controlling, etc. Compared with centralized clustering technology, the decentralized cluster has many advantages such as no single point bottleneck, no single point of failure, more flexibility [2]. However, it faces many challenges, one of which is failure detection [3]. Failure detection technologies use mechanisms such as heartbeat and timeout to provide failure sensing and troubleshooting approaches for the clusters and further make them failure tolerable [4]. Different with centralized clusters, in decentralized clusters there is no fixed supervisor who is responsible for failure detection and troubleshooting, leading to a more complicated failure detecting problem.

Existing failure detection methods [5–12] are designed to detect completely unreachable nodes, e.g., died or disconnected. A commonly used failure detection approach in a cluster is that, monitors estimate the state of each node based on the ϕ FD each node in the cluster is monitored by a set of other nodes. The monitor nodes send heartbeat requests to the target node and expect for heartbeat responses to obtain the link state between monitor and its target. If ϕ FD is adopted, monitors assume that the interval of heartbeat responses obey normal distribution. Not receiving a heartbeat response within an expected interval, they start suspecting the target to be unreachable. Then, they use gossip protocol [13] to spread the unreachability event to the rest of nodes. However, there are two main drawbacks of this kind of implementation. First, the existing failure detectors [5–12] cannot correctly detect a partially working node, e.g., a node which randomly drops packets. In this case, different monitors may give out different detection results on this node. Second, because gossip is a weak

consistency protocol, it cannot properly deal with conflicts on reachability state, which is possible to lead to the corruption of the system. Due to what we have tested, one certain malfunction breaks a cluster with a size of 10 into pieces easily.

In this paper, we analyze the causes of the problem of the current work. As the ϕ FD and other failure detection mechanisms proposed in [5–12] are not suitable for the scenario of stochastic packet loss, we propose an algorithm that can estimate the severity of packet loss of a link between two nodes based on the statistical information of TCP protocol, the round-trip-time (RTT). Further, we propose a model that can sense a node's healthy status from anywhere of the cluster without the limitation of any node to be always reachable, choose a unique leader without election process and make more reliable decisions about removing faulty nodes.

The contribution of this paper can be summarized as follows.

- First, we proposed a model to formulate the problem. Instead of modeling a link state to be reachable or unreachable, we model the link state to be healthy or unhealthy considering about unstable link. With a graph representing nodes and link states, the faulty nodes are found by solving a max clique problem(MCP). Moreover, we discuss the transitivity of the link state. For scenarios where link states have transitivity, we simplified the NP-Complete MCP to a linear complexity problem; For other scenarios, we proposed a square complexity heuristic algorithm which can find a maximal clique.
- Second, we proposed a data driven algorithm that solve the reliability issue in this specific case. Our algorithm uses an evaluation model to evaluate the link state basing on data from application layer. Basing on the evaluation results, the decision model takes care of the leader uniqueness issue and infers the faulty nodes.
- Finally, extensive simulation results demonstrate that our approach is highly adaptable. Basing on statistical data, the F1 score of the link evaluation method reaches more than 90%. And our implementation makes a real-world product stably run for more than a week while some of the the packet loss failure is injected in some of the nodes.

The rest of the paper is structured as follows. In Section 2, we introduce the related work in failure detection and troubleshooting on decentralized clusters. In Section 3, we present the formal model of our problem. In Section 4, we introduce our PingBased algorithm for enhancing the availability of decentralized cluster. In Section 5, we extensively evaluate the performance of our proposed algorithms compared with existing algorithms.

2. Related Work

2.1. Akka Cluster

A widely-used and well-recognized solution to the problem of failure detection and troubleshooting in decentralized cluster is a framework named Akka [14]. In Akka, each node is monitored by a number of other nodes with the technology of the ϕ FD. Each node in the cluster holds a reachability table and uses gossip protocol to keep consistent with other nodes. If one monitor detects that some node is faulty, it announces this event to the rest of the nodes by updating its own reachability table and make it consistent with the rest of nodes using gossip protocol. If a member is marked as faulty and has been broadcasted to all the other nodes, these nodes will then determine whether themselves shall be responsible for troubleshooting with a non-electoral leader determination algorithm. If different monitors have different outcome on whether a node is faulty, a pessimistic algorithm is adopted that nodes are only treated to be healthy if all its monitors say it is healthy, which in other words if any one of its monitors announces its faulty it will be treated as unhealthy.

2.2. Other Link Failure Detecting Algorithms

Besides ϕ FD, there are other algorithms aiming at detecting link failures. Analogous to ϕ FD, Xiong et al. [9] and Liu et al. [11] assume that the interval of heartbeat responses follows exponential

distribution or Weibull distribution and calculate the probability of time interval between the current time and the time of last heartbeat response. If the probability is lower than a threshold, the monitor suspects the monitored node. Tomsic et al. [10] uses two windows with different sizes to collect the intervals of heartbeat messages. By comparing the current time and a predicted next receipt time calculated based on these two windows, the monitor decides whether the remote node is reachable or not. Turchetti et al. [12] proposes an IFDS framework which can handle multiple concurrent applications with different QoS requirements, whose purpose is different from us.

2.3. Packet Loss Measurement

Packet loss measurement of TCP has been studied in a number of works [15–17]. Sommers et al. [15], Wu et al. [16] predict the packet loss rate with implementations in routers where they can acquire the low level sequence and acknowledge numbers of the TCP/IP stack which cannot be obtained by applications. Basso et al. [17] provide a application layer estimation on packet loss rate. However, it assumes the RTT is a constant. In addition, it mainly aims at end users who download stream from a remote server, which is different from our work.

2.4. Other Works About Fault-Tolerance

Besides the solution above, there are also other works that focus on fault-tolerance in related fields. For example, Sun et al. [18], Cerulli et al. [19], Yim et al. [20] mainly aim at failure-tolerance in a more specific area of sensor networking. R. Şinca et al. [21] focuses on digital systems and implements fault-tolerant mechanisms on the hardware field. However, targets of these works are different from our work.

3. System Overview and Problem Formulation

3.1. System Overview

In this section, we give an overview of our system of sensing healthy status of nodes in decentralized cluster based on Akka. Figure 1 shows the system overview of decentralized cluster services such as Akka cluster. Specifically, this system manages its members by maintaining a globally consistent member state table. To keep globally consistent, each node uses the Gossip Protocol [22] to repeatedly replicate its state to a randomly selected neighbor. To detect and handle node failure, each node in the cluster implements a heartbeat based failure detector to detect the reachability to some node in its neighbor. If one detects a failure, it will mark this node in its own member state table and try to gossip it to the rest of nodes in the cluster. Then a temporarily selected leader will handle this issue. We next make an expression of these processes in detail.

3.1.1. Gossip Based Membership

In order to correctly cooperate with other nodes in the cluster, each member in the cluster holds a table that contains the states of all the members in the cluster and uses Gossip protocol to make this table globally consistent. The status of the member consists of two elements, i.e., working state and reachability state. These two elements stand for whether the node is working and whether the node can be reached, respectively. To make the state globally consistent, a node periodically exchanges its state to a random neighbor. If their exchanged states are different, the node with state of older version will update its state to the newer version. To add or remove a member, a node can simply modify its member table. After that, it gossips its member states to other nodes. When the member state meets the consistency, this add or remove action is finished.

3.1.2. Failure Detection

Failure detection mechanism is used in the system to detect link failure, which further makes the cluster aware of node failures. To detect failure each node implements a failure detector. They use

heartbeat or other mechanisms to keep monitoring a number of remote nodes selected by a specific principle. If a failure is detected, the monitor node will immediately mark that node by setting the reachability status of it to be Unreachable and use gossip protocol to ensure the entire cluster finally noticing this issue. Unlike the ϕ FD adopted by Akka, the failure detection mechanism in our work has an extra feature of estimating the severity of stochastic packet loss.

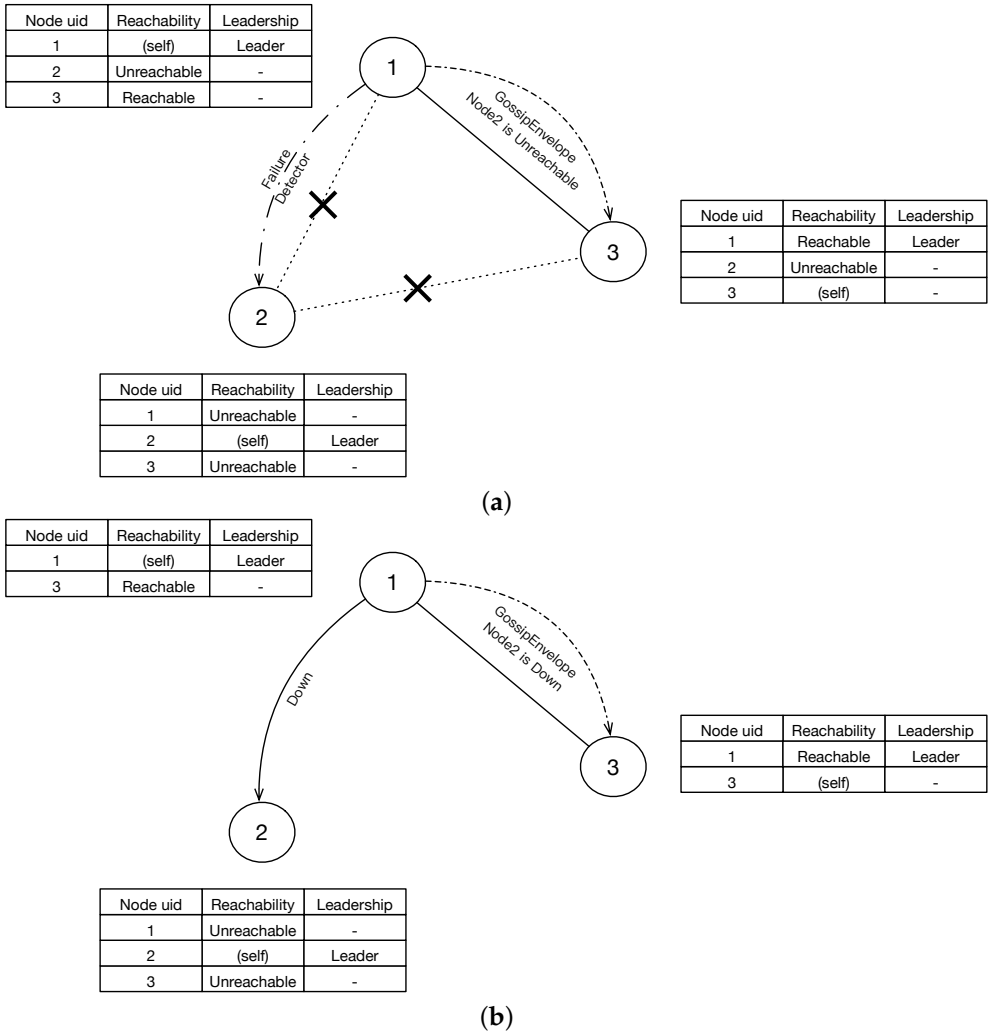

Figure 1. System overview of decentralized clusters that use gossip consensus protocol to manage members. (**a**) shows the member states, leadership and gossip process when detects a failure; (**b**) shows the down process of a leader.

3.1.3. Leadership

If an arbitrary node could decide whether a new node can join or leave, i.e., insert or remove node into or from the global membership table, it is possible to cause problems, e.g., difficulties in consistency of membership or logical issue in application. For example, if an arbitrary node could decide so, this node can do the leaving action by directly removing itself from the its own member table and gossip this table to the cluster. However, before the member table is globally consistent, some other members may not notice this leaving action and keep communicating to the leaving node. This may cause further issues like logical confusion. To avoid these problems, a temporary leader is selected to deal with actions affecting cluster's size. For adding or removing a node to or from the cluster, the leader does this action gracefully with the following steps. First, instead of suddenly adding or removing a member from the cluster, the leader sets an intermediate working state to this

member. Then the leader gossips its state to the cluster. After that, the leader waits until the member state is converged. Converged state means that the member state is consistent in the cluster so that all the members are conscious about the further action of this leaving or joining member. When the leader confirms the consistency of the intermediate state, it finally does the action of inserting or removing the member to or from the member table. The leader selection should be non-electoral to avoid being centralized. In Akka, the process is that a node considers a reachable node with the smallest unique identity to be de leader. The unique identity can be, for example, IP address and port. If the member states of all the members are consistent, it is expected that only one leader exists in the cluster at a time. And because the leader actions are only done when the leader obverses that the member state is converged, all the members will keep pace with the leader. In this way, the member state can always be easy to converge. Furthermore, the application can easily get the message of a member being joining or leaving. Thus, this approach is demonstrated to be a reliable way for changing cluster size.

3.1.4. Downing

The leader has to deal with one more case. If a member is marked as unreachable, the leader will stop the leader action until it is recovered or forcibly removed. This is because if the leader has no access to an arbitrary node, it believes that the member state cannot be consistent (because the leader cannot replicate its state to this unreachable node). We can see that a member being marked as unreachable will block the leader action which has a critical function on the joining or leaving behaviors of members. Down mechanism is thereby put forward to eliminate the long-term blocking issue. With this mechanism, if a leader believes that the unreachable nodes are no longer available, it will forcibly remove the unreachable nodes from the cluster so that the cluster would work as normal.

3.2. Mathematical Model

Table 1 shows the model of our system. The cluster is represented by a set of n nodes denoted by V. The set of actual network state are denoted by S_n. The state of each node and each link are represented by s_n and s_l, respectively. Each node manages a table of member states represented by the function fs. Nodes use Gossip protocol to make member states globally consistent. To select a leader, a node first selects a set of candidates basing on r_c, namely candidate rule. And it uses the function of $h(v)$ and $l(v)$ to determine a leader, where $h(v)$ is used to get a unique identity of v and $l(v)$ is used to return who is the leader. Leader removes a faulty node basing on the function of r_d, namely downing rule.

Particularly, we model the link in our system to be undirected [19,23], i.e., $s_l(v_i, v_j)$ is equal to $s_l(v_j, v_i)$. In addition, we only consider the scenario where the majority of the nodes in the cluster work,

$$V_h := \{v : v \in V, s_n(v) = Healthy\},\ |V_h| > \frac{|V|}{2} \tag{1}$$

We next talk about an important property of connectivity between nodes, i.e., transitivity. With the property of transitivity, no partial connectivity is appeared in the topology. This is to say two healthily connected end-nodes have the same link states to any other end-nodes. In traditional network this property is applicable because the endpoints do not have the ability to forward messages to others and the route protocol will eventually take care of the partially connecting issue. However, in case of topology like ad-hoc networking where endpoints are responsible for forwarding data or in case that unfair QoS is adopted, this property may not be applicable then. Therefore, we classify our algorithm into two cases divided by the applicability of this property. The transitivity can be described as the following formula:

$$\exists v_i, v_j \in V, v_i \neq v_j, s_l(v_i, v_j) = Healthy, \Rightarrow \forall v_k\ s_l(v_i, v_k) = s_l(v_j, v_k) \tag{2}$$

Table 1. Formal system model.

(a) A set V of n nodes ($n = |V|$)
(b) A set S_n. Two functions $s_l : V \times V \to S_n$, $s_n : V \times V \to S_n$. s_l maps link to link state; s_n maps node to node state.

$$S_n = \{Healthy, Unhealthy\}$$

$$s_l(v_i, v_j) = \begin{cases} Unhealthy, & \text{iff the link from } v_i \text{ to } v_j \text{ is unhealthy} \\ Healthy, & \text{otherwise} \end{cases}$$

$$s_n(v) = \begin{cases} Unhealthy, & \text{iff the node } v \text{ is unhealthy} \\ Healthy, & \text{otherwise} \end{cases}$$

An equation:

$$s_l(v_i, v_j) = s_l(v_j, v_i) \tag{3}$$

(c) A set S_R of two reachability states and a function $fs : V \times V \to S_R$ that returns the result of failure detecting.

$$S_R = \{Reachable, Unreachable\}$$
$$fs(v_i, v_j) = \text{reachability state of } v_j \text{ seen by } v_i$$

(d) A procedure *GossipTo*:

$$v_i \ GossipTo \ v_j : \ \forall v \in V \text{ let } fs(v_j, v) \leftarrow fs(v_i, v) \text{ iff } fs(v_i, v) \text{ is newer}$$

(e) A function $r_c : V \times V \to \{TRUE, FALSE\}$ used as candidate rule in leader selection process.

$$r_c(v_i, v_j) = \begin{cases} TRUE, & \text{iff } v_j \text{ can be a candidate from the perspective of } v_i \\ FALSE, & \text{otherwise} \end{cases}$$

and a relation $cand : V \to \mathbb{P}(V)$ where $v_k \in cand(v_i)$ iff $r_c(v_i, v_k) = TRUE$
(f) Two functions about non-electoral leader selection $h : V \rightarrowtail \mathbb{I}$, $l : V \to V$

$$h(v_x) = \text{unique identity of } v_x$$
$$l(v_i) = \underset{v_x \in cand(v_i)}{argmin} \ h(v_x)$$

and an integer $k, 0 < k \leqslant n$, stands for number of leaders at a time:

$$k = |\{v : v \in V, l(v) = v\}|$$

(g) A function $r_d : V \times V \to \{TRUE, FALSE\}$ used as rule of Downing. A procedure *DoDowning*

$$r_d(v_i, v_j) = \begin{cases} TRUE, & \text{iff } l(v_i) = v_i \text{ and } v_i \text{ believes } v_j \text{ is no longer vaild} \\ FALSE, & \text{otherwise} \end{cases}$$

$$v_i \ DoDowning : \text{let } V \leftarrow V \setminus \{v : v \in V, r_d(v_i, v) = TRUE\}$$

3.3. Problem Formulation

The main weakness of gossip based membership management is that once a node receives a newer version of member state from a valid sender it will merge the state into its own state with only a simple conflict avoidance logic. This influences little on a normally working cluster. However, in some cases, a problematic but valid node gossiping corrupt member state can bring severe problem to the cluster. Stochastic packet loss is a typical case of a node being problematic but valid. In this case, the

problematic node with unstable links to other nodes may mark a part of other nodes as Unreachable uncertainly if the failure detection mechanism is unreliable. Moreover, unlike network partition issue, this node still has possibility to gossip its globally incorrect member state. If the incorrect member state corrupts the cluster, it will cause at least 2 fatal problems:

1. The leader may remove normal nodes that are marked as unreachable by the problematic one if the downing rule is not reliable.
2. There may be more than one nodes assume themselves to be leaders if all the normally working nodes with smaller unique id are marked by the problematic node.

More specifically, a faulty node v_f marks a set $V_x \subseteq V \setminus \{v_f\}$ of nodes as Unreachable by the process of $\forall v \in V_x$ let $fs(v_f, v) \leftarrow Unreachale$, and GossipTo $v_j, v_j \in V \setminus V_x \setminus \{v_f\}$. Finally, the member state might converge to a result that $\forall v_i, v_j, v_k \in V$ s.t. $fs(v_i, v_k) = fs(v_j, v_k)$. The commonly used original implantations of r_c and r_d are shown in Table 2. In this case, the uniqueness of leader cannot be guaranteed. As a result, it may happen that $2 \leqslant k \leqslant n$ (recall Table 1f, k stands for the quantity of leaders). Meanwhile, nodes in the set of V_x will be removed by these leaders after a period of time regardless of their actual node states.

Table 2. Original implementation of r_c and r_d.

(a) The original implementation of function r_c:

$$r_c(v_i, v_j) = \begin{cases} TRUE, & \text{iff } fs(v_i, v_j) = Reachable \\ FALSE, & \text{iff } fs(v_i, v_j) = Unreachable \end{cases}$$

(b) The original implementation of function r_d:

$$r_d(v_i, v_j) = \begin{cases} TRUE, & \text{iff } fs(v_i, v_j) = Unreachable \text{ for a period of time and do not recover} \\ FALSE, & \text{otherwise} \end{cases}$$

From the discussion above, we can draw 4 problems. First, an approach for estimating the severity of packet loss of link must be found. Second, faulty nodes must be prevented to make any decision on changing the size of cluster. Third, the uniqueness of leader node should be guaranteed. Finally, the leader need an approach to find faulty nodes and do responsible troubleshooting. These problems can be formulated as follows:

(a) Link Estimate Problem (LEP)

Given: A local node v_i and a remote node v_j.

Problem: Find a network indicator $I_n(v_i, v_j)$ and a function $s'_l : I_n \rightarrow S_n$ such that the misrecognition rate of link state σ_l is minimized, which can be expressed as:

$$\sigma_l(v_i, v_j, I_n) := P(s'_l(I_n)! = s_l(v_i, v_j)), \min \sigma_l.$$

(b) Self Checking Problem (SCP)

Given: A local node v.

Problem: Find a node indicator $I_v(v)$ and a function $s'_n : I_v \rightarrow S_n$ such that the misrecognition rate of node state σ_n is minimized, which can be expressed as:

$$\sigma_n(v, I_v) = P(s'_n(I_v)! = s_n(v)), \min \sigma_n.$$

(c) Leader Uniqueness Problem (LUP)

Given: The node set V, the observation of connectivity from node v_i to v_j denoted by $s'_l(v_i, v_j)$.

Problem: Find a new implementation of function r_c such that at a specified time it is guaranteed that the quantity of leaders in the cluster is no more than 1, which can be expressed as $k \leqslant 1$.

(d) Faulty Nodes Determination Problem (FNDP)

Given: The node set V, the observation of connectivity from node v_i to v_j denoted by s'_l and the candidate rule r_c.

Problem: Find a new implementation of function r_d such that reliability of the removing action is maximized. A reliable removing action can be defined as removing only faulty nodes by the leader.

4. The PingBasedDown Algorithm

The PingBasedDown Algorithm is a distributed solution that helps to detect problematic nodes reliably, which therefore enhance the availability of decentralized clusters who use Gossip protocol as their main protocol to manage their membership. Each node in the cluster implements the full function of this algorithm.

In our algorithm, we first collect enough data, which can be potentially used as network indicators I_n, from application layer. After appropriately preprocessing, we bring them to the first model called Link Evaluation Model which is designed to estimate the link quality with the consideration of stochastic packet loss over TCP protocol. As for other message-based protocols such as UDP, the packet loss rate can be estimated simply using the ping-pong tests without this model so that we do not discuss them in this paper. The link evaluation results are then be used by the next model called Decision Model. It firstly evaluates the healthy status of the local and remote nodes and then determines the leadership. Finally, the node chosen as leader executes the faulty nodes selected by this model.

Figure 2 shows the overall solution of the PingBasedDown algorithm.

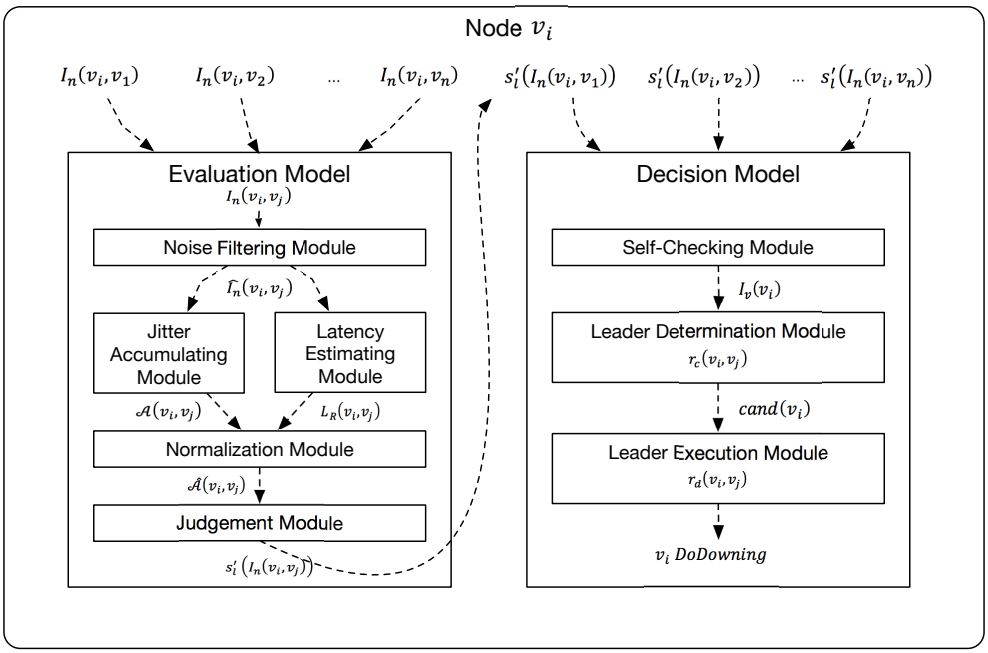

Figure 2. Algorithm overview.

In preparation for describing in detail, we define some operators on the vector and set:

Definition 1. *(FindFirst: $\mathbb{X}^m \times \mathbb{I} \to \mathbb{I}$) Giving a vector $X = (x_1, x_2, ..., x_m)$, $FindFirst(X, \alpha)$ is defined as returning the subscript of the first value in X that equals to α. For example, suppose $X = (1, 3, 5, 5, 4)$, $FindFirst(X, 5) = 2$ because x_2 is the first element in X that equals to 5.*

Definition 2. *(RemoveAt: $\mathbb{X}^m \times \mathbb{I} \to \mathbb{X}^{m-1}$) Giving a vector $X = (x_0, x_1, x_2, ..., x_m)$ and a subscript $i, 0 \leqslant i \leqslant m$, $RemoveAt(X, i) = (x_0, x_1, x_2, ..., x_{i-1}, x_{i+1}, ..., x_m)$. For example, suppose $X = (1, 3, 5, 5, 4)$, $RemoveAt(X, 3) = (1, 3, 5, 4)$.*

Recall Section 3.3, to solve our problem we should first find a network indicator $I_n(v_i, v_j)$, which is sensitive to packet loss and hence has the ability to estimate the quality of link.

After the analysis of the collected data over TCP protocol with different packet loss rate, we find that the round-trip-time (RTT) of a TCP message is especially sensitive to stochastically packet loss. Here the RTT of a RPC message means time between a certain kind of message and its reply. We then fetch this feature and use it as the network indicator.

To construct the input, i.e., $I_n(v_i, v_j)$, of our model, the node v_i keeps collecting the most recent N_w groups of timestamps of communication records to v_j. The timestamp group consists of the sending timestamp of a message and the receiving timestamp of its reply. We then calculate a RTT by subtract the two timestamps. Consider that it may take some time for a remote node v_j to process some of the messages, we should subtract the processing durations from corresponded RTTs. We denote $T_r(v_i, v_j)$ to be the vector of collected RTT from v_i to v_j, T_p to be a vector of processing delay corresponding to the vector of T_r. Then, the notations of T_r, T_p and the indicator I_n can be expressed as follows.

$$\begin{aligned} T_r(v_i, v_j), T_p(v_i, v_j) &: V \times V \to \mathbb{R}_+^{N_w}, \\ T_r(v_i, v_j) &= \text{The most recently collected } N_w \text{ groups of round-trip-times}, \\ T_p(v_i, v_j) &= \text{The processing time of messages corresponding to } T_r(v_i, v_j), \\ I_n(v_i, v_j) &= T_r(v_i, v_j) - T_p(v_i, v_j). \end{aligned}$$

A simplified approach to construct the input is to collect the RTTs of messages which are supposed to be replied immediately, e.g., heartbeat messages. In this case, the indicator can be $I_n(v_i, v_j) = T_r(v_i, v_j)$.

4.1. Link Evaluation Model

Link evaluation model is proposed to evaluate the link quality and solve the LEP. Specifically, this model provides an implementation of function s'_l. In addition to the two original states, a fuzzy state Pending is introduced to avoid any arbitrary judgments. This model consists of the following five modules and Table 3 shows the parameters of this model. First is Noise filtering module, which filters the noise brought by applications from the collected RTTs. The filtered input then goes to the next module named Jitter accumulating module to quantify the jitter. The filtered input also goes to the module named Latency estimating module, which estimates the pure latency, i.e., latency without processing or retransmitting delay, of the link. The quantified jitter and the estimated latency then go through the Normalization module to calculate a normalized result. Finally, the result is compared with two thresholds to evaluate the state of the link. Next, we make detailed description on these modules.

Table 3. Parameters of Link Evaluation Model.

Notation	Parameter Name	Description
N_w	History size	How many groups of rtt history we use as the basis.
F_S	Filter strength	Indicates the strength of filtering, e.g., the proportion of noise in the records.
L_{pos}	Latency positioning factor	Indicates rate of pure latency participate in calculating among history of RTTs.
T_{alert}	Alert threshold	If the normalized accumulated value is higher than this threshold, mark the link as Unhealthy
T_{safe}	Safe threshold	If the normalized accumulated value is lower than this threshold, mark the link as Healthy

4.1.1. Noise Filtering Module

Since our system works upon application layer, the input of this model $I_n(v_i, v_j)$ is expected to contain noises brought by the application, e.g., garbage collection or thread scheduling process. This module is used to preprocess the input to eliminate the impact of noises. For different scenarios, different implementations of noise filtering modules can be implemented, e.g., removing a part of highest numbers from collected RTTs. Algorithm 1 shows an implementation of noise filtering process. We denote the function of this process as follows,

$$NF : \mathbb{R}^{N_w} \rightarrow \mathbb{R}^{[N_w \times (1-F_S)]} \tag{4}$$

Then the output $\widehat{I_n}(v_i, v_j) = NF(I_n(v_i, v_j))$ will be used in next steps.

Algorithm 1: Simple noise filtering algorithm.

Input: a vector of collected RTTs denoted by $Rtts$, $Rtts \in \mathbb{R}_+^{N_w}$
Initialize:
  $\mathcal{R}_f \leftarrow Rtts$
  $k \leftarrow [N_w \times F_S]$
  $i \leftarrow 0$
while $i < k$ **do**
  $m_v \leftarrow \max \mathcal{R}_f$
  $i_m \leftarrow FindFirst(\mathcal{R}_f, m_v)$
  $\mathcal{R}_f \leftarrow RemoveAt(\mathcal{R}_f, i_m)$
  $i \leftarrow i + 1$
Output: $\mathcal{R}_f$

4.1.2. Jitter Accumulating Module

Calculating the variance of data is a commonly used approach to measure the jitter. However, this method is inaccurate in some cases. For example, the two vectors $d_1 = (1, 1, 1, 100, 100, 100)$ and $d_2 = (1, 100, 1, 100, 1, 100)$ have the same variance, but what we want is that the jitter of d_2 is higher than that of d_1. Therefore, in this module, we quantify the jitter of $\widehat{I_n}(v_i, v_j)$ by accumulating the quantified variation rate. Let $\mathcal{A}$ denote the quantification result. We quantify the jitter by the following steps:

Giving a vector of numbers $X \in \mathbb{R}^{N_w}$, we use first-order difference of this vector ΔX to extract the rate of variation, i.e., jitter. Then we obtain the quantified jitter by accumulating the absolute value of ΔX. The output of this module $\mathcal{A}$ can then be represented as follows:

$$\mathcal{A} : \mathbb{R}^{N_w} \rightarrow \mathbb{R},\ \mathcal{A}(X) = \sum |\Delta X| \tag{5}$$

4.1.3. Latency Estimating Module

In order to make the algorithm adapt to different levels of latency, this module is proposed to estimate the pure latency of the link. The pure latency denoted by L_R means how long a RTT of the message is without triggering the retransmission mechanism. Through a normal link from v_i to v_j, L_R is expected to be: $L_R(v_i, v_j) = \mathcal{E}\left(\widehat{I_n}(v_i, v_j)\right)$, where $\mathcal{E} : \mathbb{R}^{N_w} \to \mathbb{R}$ is the operator of mathematical expectation. However, when messages transmit through an abnormal link with stochastic packet loss, the RTT sometimes may be much longer than the pure latency because of the retransmit mechanism. Thus, we introduce an approach. It cuts off the bigger part of the collected RTTs which are supposed to be caused by retransmission process. Then it calculates the average value of the rest, i.e., the smaller part, of RTTs which are supposed to be transmitted without packet loss. This average value is used as the estimate of pure latency. The cut off action is similar to that of the noise filtering algorithm presented in Algorithm 1 with the replacement from F_S to L_{pos}.

4.1.4. Normalization Module

From the algorithm of the jitter accumulating module, we can find that the value of the accumulated result has a strong correlation with the length of the vector, denoted by $\dim(\mathcal{A})$, and the level of the link latency. This brings difficulty on the judgement of link quality. The Normalization module adjusts this value to a notionally common non-dimensional scale using the formula as follows,

$$\hat{\mathcal{A}} = \frac{\mathcal{A}}{L_R \times \dim(\mathcal{A})} \tag{6}$$

With the help of this module, no matter what level of link latency is and how long the vector is, the evaluation result for links with same packet loss rates should be approximately within a same range.

4.1.5. Judgement Module

Recall the very first of this section, we define 3 states of link quality:

- Healthy, which stands for normally working link without packet loss;
- Unhealthy, it stands for abnormal link with packet losses;
- Pending, which stands for fuzzy link which may need further detection.

To determine which state the link should be, we compare the $\hat{\mathcal{A}}$ with two thresholds, namely safe threshold and alert threshold, denoted by T_{safe} and T_{alert}, respectively. In addition, the link status is determined by the following equation:

$$I_n(v_i, v_j) = \begin{cases} Unhealthy, & \hat{\mathcal{A}} \geqslant T_{alert}, \\ Pending, & T_{alert} > \hat{\mathcal{A}} > T_{safe}, \\ Healthy, & \hat{\mathcal{A}} \leqslant T_{safe}. \end{cases} \tag{7}$$

Two parameters of T_{safe} and T_{alert} determine the sensitivity of Link Evaluation Model on link failures. A higher T_{alert} makes the model more stable and decreases the false alarm rate when working on noisy networks. However, an exorbitant T_{alert} also makes the model hard to detect a link failure. Moreover, when the normalized accumulated jitter $\hat{\mathcal{A}}$ is lower but very closed to T_{alert}, it indicates that the link quality is fuzzy. To make our model more robust, we must prevent giving a Healthy mark on fuzzy links. Therefore, we propose the T_{safe} threshold. A lower T_{safe} makes the model give a Healthy mark of a link more strictly. In the datacenter environment where the nodes in the cluster are physically closed to each other, we believe that the jitter rate of latency there is low and therefore a low T_{safe} shall be set. As for other scenarios such as cloud services, T_{safe} shall be set to a higher value to make the model properly working. These two thresholds can be determined either statically by empirical values or dynamically by adaptive algorithms. In the simulation, we set the thresholds statically based on a long statistical data.

4.2. Decision Model

With the link evaluation results, the decision model is proposed to solve the following three problems, first is whether a node itself is healthy(SCP), second is whether a node is the leader who is responsible for removing faulty nodes(LUP), third is which nodes are faulty(FNDP).

We denote the cluster (nodes and its links) to be a simple undirected graph $G = (V, E)$, where $E = \{(v_i, v_j) : \forall v_i, v_j \in V, s_l(v_i, v_j) = Healthy\}$. As shown in Figure 3, with the help of evaluation model, a node v_i can build a group of subgraphs $G'(v_i) = (V, E'(v_i))$ and $\overline{G}'(v_i) = (V, \overline{E}'(v_i))$ with the result of Evaluation model. Similar to the relationship between s_l and s'_l, E' (and also $\overline{E}'$) is the observation of E. If an edge is in the set E', the evaluation result of this link is Healthy. As for edges existing in $\overline{E}'$, their evaluation results are Unhealthy. If an edge (v_i, v_j) is neither in $E'(v_i)$ nor in $\overline{E}'(v_i)$, its evaluation result is Pending. Formally, E' and $\overline{E}'$ are defined as follows:

$$E'(v_i) := \{(v_i, v) : \forall v \in V, s'_l(I_n(v_i, v)) = Healthy\}, \tag{8}$$

$$\overline{E}'(v_i) := \{(v_i, v) : \forall v \in V, s'_l(I_n(v_i, v)) = Unhealthy\}. \tag{9}$$

Next, we present some definitions about the node state.

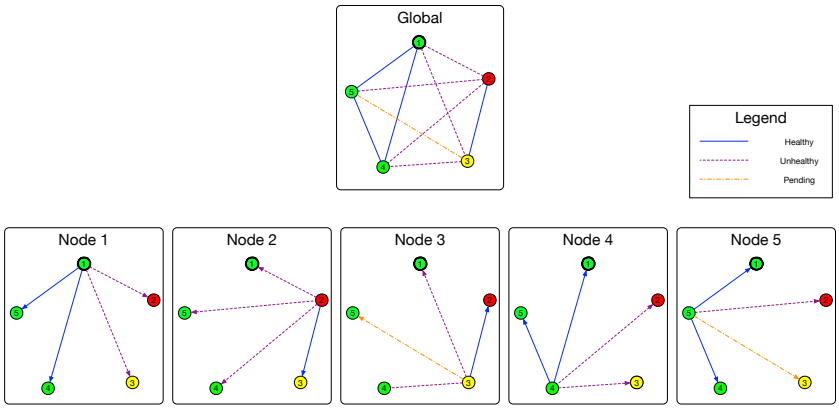

Figure 3. Example of global connectivity graph and sub connectivity graphs. In this example, $E'(v_3) = \{(v_3, v_2)\}$ and $\overline{E}'(v_3) = \{(v_3, v_1), (v_3, v_4)\}$

Definition 3. *Healthy node set. W is the healthy node set if and only if the following 3 conditions are satisfied: (1)* $\forall v_1, v_2 \in W$, $(v_1, v_2) \in E$; *(2)* $|W| \geqslant [\frac{V}{2}]$; *(3)* $\nexists v \in V$ *s.t.* $\forall v_h \in W, (v, v_h) \in E$.

Definition 4. *Healthy node. A node v is Healthy if and only if* $v \in W$.

Lemma 1. *When a cluster is normally working, if a node* v_i *is healthy, the degree of* v_i *in G must be greater than* $[\frac{|V|}{2}] - 1$.

$$(1), s_n(v_i) = Healthy \Rightarrow deg(v_i) > \left[\frac{|V|}{2}\right] - 1 \tag{10}$$

Proof. Let $V_h := \{v : v \in V, s_n(v) = Healthy\}$, $V'_h := V_h \setminus \{v_i\}$. Because of (1), we have $|V_h| > \left[\frac{|V|}{2}\right]$, and according to the definition of V'_h, we have:

$$|V'_h| > \left[\frac{|V|}{2}\right] - 1 \tag{11}$$

With Definition 4, we have $deg(v_i) = |V'_h| - 1$. Combining with (11), we have $deg(v_i) > [\frac{|V|}{2}] - 1$. □

Lemma 2. *A node v_i is unhealthy, if and only if the degree of v_i in G must lower than $[\frac{|V|}{2}]$.*

$$(1), s_n(v_i) = Unhealthy \Leftrightarrow deg(v_i) < [\frac{|V|}{2}] \tag{12}$$

From Lemma 1 we see that the condition that $deg(v_i) > [\frac{|V|}{2}] - 1$ is only the necessary but not the sufficient condition of that v_i is healthy. However, these nodes with degree greater than half of the cluster size are also potentially healthy, which may need further determination. Thus, we make a new definition with this kind of nodes to be PendingHealthy.

Definition 5. *If the degree of a node v in G is greater than $[\frac{|V|}{2}] - 1$, we say that it is PendingHealthy.*

$$deg(v_i) > [\frac{|V|}{2}] - 1 \Leftrightarrow v_i \text{ is } PendingHealthy \tag{13}$$

Particularly, when the transitivity is applicable, if a node is PendingHealty, it must be Healthy.

Theorem 1. *When the transitivity is applicable, if a node is PendingHealthy, it is Healthy. This can be expressed as:*

$$(1), (2), deg(v_i) > [\frac{|V|}{2}] - 1 \Rightarrow s_n(v_i) = Healthy \tag{14}$$

Proof. We let $V'_h := \{v : v \in V, s_l(v_i, v) = Healthy\} \cup \{v_i\}$. For any two remote nodes v_j and v_k in V'_h, according to the transitivity (2), the link quality of v_j to v_k can be inferred by the local node v_i, which can be expressed as:

$$\forall v_j, v_k \in V'_h \setminus \{v_i\},\ s_l(v_j, v_k) = Healthy \tag{15}$$

Because the local node v_i is PendingHealthy, the degree of v_i is greater than $[\frac{|V|}{2}] - 1$ so that the size of V'_h is greater than half of the cluster's size: $|V'_h| > [\frac{|V|}{2}]$. (15) means that the nodes in $|V'_h|$ are fully connected. Combine with Definition 4, the state of v_i is Healthy, i.e., $s_n(v_i) = Healthy$. □

We use the following 3 modules to solve the problem of SCP, LUP and FNDP. They are used to check the state of local node, ensure the uniqueness of leader, construct a global G' as an approximation of graph G and remove faulty nodes based on G', respectively.

4.2.1. Self-Checking Module

At any time, a faulty node should not be selected as the leader. However, the evaluation model or other failure detection mechanisms cannot sense which side, i.e., whether themselves or their peers, is faulty. We thereby design this module to do self-checking and if one's self-checking procedure indicates that itself is faulty, it will abandon all the next steps and report this issue to the upper applications. More specifically, this module proposes the indicator I_v and the function of s'_n.

With the basis of Lemma 1, Lemma 2 and Definition 5, we come up with the idea of this module that, if the majority of the nodes in the cluster announce that $v_f (v_f \in V)$ is the faulty one using the evaluation module, v_f should be unhealthy. Although it is possible that all the announcers are unhealthy, however, in that case we can say that most of the nodes in the cluster have failed so that the cluster is totally out of function and it would be meaningless to discuss the reliability and availability. On the contrary of the condition of unhealthy, if the majority ones believe that v_w is healthy, v_w should be healthy although it is actually PendingHealthy which we have already discussed about.

This idea can then be described as:

$$\begin{cases} I_v(v_i) := \{v : v \in V \setminus \{v_i\},\ (v, v_i) \in E'(v)\}, \\ \overline{I_v}(v_i) := \{v : v \in V \setminus \{v_i\},\ (v, v_i) \in \overline{E}'(v)\}, \\ s'_n(I_v) = \begin{cases} Unhealthy, & |\overline{I_v}| > [\frac{|V|}{2}], \\ Healthy, & |I_v| > [\frac{|V|}{2}] - 1, \\ Pending, & \text{otherwise.} \end{cases} \end{cases} \tag{16}$$

Unfortunately, the self-checking process is performed in v_i, who has no knowledge about $E'(v)$ and $\overline{E}'(v)$, $\forall v \in V \setminus \{v_i\}$. We thereby consider that $s_l(v, v_i) = s_l(v_i, v)$. Because $s'_l(I_n(v, v_i)) = s_l(v, v_i)$ is expected, $s'_l(I_n(v_i, v)) = s'_l(I_n(v, v_i))$ is also expected due to transitivity. Combine with (8) and (9), we have the following theorem:

Theorem 2. *If* $(v_i, v_j) \in E'(v_i)$ *then* $(v_j, v_i) \in E'(v_j)$ *and if* $(v_i, v_j) \in \overline{E}'(v_i)$ *then* $(v_j, v_i) \in \overline{E}'(v_j)$.

With Theorem 2, the function of s'_n can be easily calculated locally because (16) can be converted to:

$$\begin{cases} I_v(v_i) = \{v : v \in V \setminus \{v_i\}, (v_i, v) \in E'(v_i)\}, \\ \overline{I_v}(v_i) = \{v : v \in V \setminus \{v_i\},\ (v_i, v) \in \overline{E}'(v_i)\}. \end{cases} \tag{17}$$

If the self-checking result is healthy, this node should go forward to leader determination module. Otherwise, the node may suffer from network failure and it should handle this issue. If the result is pending, the node may do nothing but wait for more reliable link evaluation results.

4.2.2. Leader Determination Module

This module is to make healthy nodes find the cluster leader without election, i.e., find the function r_c. Recall Table 1 (f), each node is given a unique id by function $h(v)$. With the help of this function, the main idea of this module can be described as choosing the node from the healthy set with the minimal unique id to be the leader. Therefore, we define the basic candidate rule r_c to be:

$$r_c(v_i, v_j) = \begin{cases} TRUE, & iff\ s'_n(I_v(v_j)) = Healthy, \\ FALSE, & otherwise. \end{cases} \tag{18}$$

However, in this case of the basic candidate rule, we can find that one v_i must obtain all remote nodes' healthy states, i.e., $s'_n(I_v(v_j))$ for all $v_j \in V \setminus \{v_i\}$, which cannot be directly acquired locally. For this reason, an approach is needed to get or infer the status of remote nodes. Here we need to consider about the transitivity. Recall Section 3.2, this property is applicable in most cases but do have exceptions. Hence, we propose two different approaches classified by the applicability of this property.

The first approach is to infer healthy states of remote nodes. In the most common conditions that transitivity is applicable, the two following theorems that can be proved:

Theorem 3. *When transitivity is applicable, for an arbitrary node* v_t*, if it has a healthy link that connected to a healthy node* v_s*,* v_t *must be healthy. This can be formulated as:*

$$\forall v_s, v_t \in V : \left\{ \begin{array}{c} s_n(v_s) = Healthy \\ s_l(v_s, v_t) = Healthy \end{array} \right\} \Rightarrow s_n(v_t) = Healthy$$

Proof. Because $s_n(v_s) = Healthy$, according to Definition 4, we have:

$$\exists V'_h \; s.t. \begin{cases} |V'_h| > [\frac{|V|}{2}] - 1, \\ \text{nodes in } V'_h \text{ are fully connected}, \\ v_s \in V'_h. \end{cases} \tag{19}$$

Combine with (19), transitivity (2), and the fact that $s_l(v_s, v_t) = Healthy$, it can be inferred that $\forall v \in V'_h \setminus \{v_t\}, s_l(v, v_t) = Healthy$. Thus, we have:

$$deg(v_t) > [\frac{|V|}{2}] - 1 \tag{20}$$

According to Theorem 1 and (20), the state of v_t can be demonstrated: $s_n(v_t) = Healthy$. □

Lemma 3. *(With transitivity) For three nodes v_i, v_j and v_k, if the link state from v_i to v_j is Healthy and that from v_i to v_k is Unhealthy, it can be inferred that the link state from v_j to v_k is Unhealthy. This lemma can be formulated to:*

$$\exists v_i, v_j, v_k \in V \; s.t. \left\{ \begin{array}{c} s_l(v_i, v_j) = Healthy \\ s_l(v_i, v_k) = Unhealthy \end{array} \right\} \Rightarrow s_l(v_j, v_k) = Unhealthy$$

Theorem 4. *(With transitivity) For two nodes v_s and v_t, if the state of v_s is Healthy, and the link between v_s and v_t is Unhealthy, it can be inferred that the state of v_t is Unhealthy.*

$$\exists v_s, v_t \in V \; s.t. \left\{ \begin{array}{c} s_n(v_s) = Healthy \\ s_l(v_s, v_t) = Unhealthy \end{array} \right\} \Rightarrow s_n(v_t) = Unhealthy$$

Theorem 4 can be proved similar to the proof of Theorem 3.

Proof. According to Lemma 3 and (19), because the link status $s_l(v_s, v_t) = Unhealthy$, we have inferred that $\forall v \in V'_h : s_l(v, v_t) = Unhealthy$, which also can be expressed as $deg(v_t) < [\frac{|V|}{2}]$. Combine with Lemma 2, we can conclude that $s_n(v_t) = Unhealthy$. □

With Theorems 3 and 4, the state of a remote node can be easily inferred from the link evaluation results. In short, when the transitivity is applicable, if the state of a link starts with a healthy node is healthy, the destination remote node is in state of healthy. Hence the candidate rule r_c can be redefined to:

$$r_c(v_i, v_j) = \begin{cases} TRUE, & iff \; s'_l(I_n(v_i, v_j)) = Healthy, \\ FALSE, & otherwise. \end{cases} \tag{21}$$

The second approach is fetching healthy states from remote nodes. In the condition that the transitivity is not applicable, we have to fetch all the partial topologies from remote nodes and combine them to a global topology. This helps us to choose a unique leader in this module and further help to do execution in the next module.

Recall the very first of this section, partial topology in node v_x are denoted by a group of directed graphs $G'(v_x) = (V, E'(v_x))$ and $\overline{G}'(v_x) = (V, \overline{E}'(v_x))$ and the global topology is denoted by an undirected graph $G = (V, E)$. For an arbitrary healthy node v_i, the target of this module is to fetch the remote partial $G'(v)$ for $v \in V - \{v_i\}$, and combine these $G'(v)$ to an undirected graph $G' = (V, E')$ which is expected to be equal to G. When a node successfully constructs G', it will be able to calculate the states of nodes using the approach provided in Self-Checking module. The candidate rule can then be defined as:

$$r_c(v_i, v_j) = \begin{cases} TRUE, & deg_{G'}(v_j) > [\frac{|V|}{2}] - 1, \\ FALSE, & otherwise. \end{cases} \tag{22}$$

To introduce our approach in detail, we here make definitions on Healthy observation set and Unhealthy observation set of nodes.

Definition 6. *Healthy observation set of an arbitrary node v_x is the set of nodes that the link between them and v_x is healthy, which can be expressed as $S_h(v_x) := \{v : v \in V \setminus \{v_x\}, s_l(v_x, v) = Healthy\}$.*

Definition 7. *Unhealthy observation set is defined as v_x: $\overline{S_h}(v_x) := \{v : v \in V \setminus \{v_x\}, s_l(v_x, v) = Unhealthy\}$.*

For implementation, we also use $s'_l(I_n(v_i, v_j))$ to approximate $s_l(v_i, v_j)$ in order to obtain the observed S_h and $\overline{S_h}$ in each node.

To combine the partial topologies into a global topology, we have to deal with the following issues (suppose the local node is v_l). First, v_l fetches partial topologies from the nodes in $S_h(v_l)$ directly. Then, v_l asks for partial topologies of unreachable nodes, i.e., nodes in $\overline{S_h}(v_l)$, with the help of nodes in $S_h(v_i)$. While fetching partial topologies, handle the ask timeout. While combining the partial topologies, check for and resolve the conflict state of a certain edge from two sides of nodes. Finally, give the combination result.

Algorithms 2–4 shows the full workflow of this module.

Algorithm 2 shows the workflow of fetching and combining process. This process is started by the initiator who tries to acquire the global topology. It first initializes a pair of mutable graphs $(\mathcal{G}', \overline{\mathcal{G}}')$, which will eventually hold the combination result, i.e., global graph; a mutable set V_{seen}, which indicates whose sub connectivity graphs have been combined into the the intermediate $\mathcal{G}'$ and $\overline{\mathcal{G}}'$, an immutable Healthy observation set S_h. For all nodes in $S_h \cap V_t$, it sends a *AskTopo* RPC to these nodes. If a *ReplyTopo* is replied, try to merge the replied graph $G'(v_r), \overline{G}'(v_r)$ with $\mathcal{G}', \overline{\mathcal{G}}'$. If merging process succeeded, combine the returned seen set V'_{seen} with the local seen set V_{seen}. If merging process failed, stop the fetching and combining process immediately and report the conflict issue. If no *ReplyTopo* is replied, mark the edge from the local node to this remote node as Unhealthy and also add this remote node into seen set.

Algorithm 4 shows the workflow when a node receives a *AskTopo* RPC. If the sender of this RPC asks for aid to grab sub connectivity graphs from other nodes in $\overline{S_h^r}$ (because the sender cannot connect to these nodes), the receiver will invoke FetchAndCombine and set the parameter V_t to be $S_h \cap \overline{S_h^r}$. If $V_t = \emptyset$, the FetchAndCombine will do nothing but return its own sub graph. After FetchAndCombine process is completed, the receiver will pack the combine result into ReplyTopo message and send back to the sender of AskTopo RPC.

Algorithm 2: Fetching and combining algorithm. FetchAndCombine($V, v_l, G', \overline{G}', V_t, V_{aid}$).

Input:
- the node set V
- local node v_l
- its sub connectivity graph $G'(v_l) = (V, E'(v_l))$, $\overline{G}'(v_l) = (V, \overline{E}'(v_l))$
- node set V_{aid} that need others help to fetch.

Initialize:
- $\mathcal{E}' \leftarrow E'(v_l)$, $\mathcal{G}' \leftarrow (V, \mathcal{E}')$
- $\overline{\mathcal{E}}' \leftarrow \overline{E}'(v_l)$, $\overline{\mathcal{G}}' \leftarrow (V, \overline{\mathcal{E}}')$
- $V_{seen} \leftarrow \{v_l\}$
- $S_h \leftarrow V \setminus \overline{E}'(v_l)$

for $v_r \in S_h \cap V_t$ **do**
- Send a RPC AskTopo(v_l, V_{aid}) to v_r
- **if** *Got the ReplyTopo*($G'(v_r), \overline{G}'(v_r), V^r_{seen}$) **then**
  - $(\mathcal{G}', \overline{\mathcal{G}}', status) \leftarrow Merge(\mathcal{G}', \overline{\mathcal{G}}', V_{seen}, G'(v_r), \overline{G}'(v_r), V^r_{seen})$
  - **if** $status \neq success$ **then**
    - **return** $(\emptyset,\emptyset), (\emptyset,\emptyset), conflict$
  - **else**
    - $V_{seen} \leftarrow V_{seen} \cup V^r_{seen}$
- **else**
  - $V_{seen} \leftarrow V_{seen} \cup \{v_r\}$
  - $\overline{\mathcal{E}}' \leftarrow \overline{\mathcal{E}}' \cup (v_l, v_r)$

return $\mathcal{G}', \overline{\mathcal{G}}', success$

Algorithm 3: Merging algorithm. Merge($G_1, \overline{G}_1, V_1^{seen}, G_2, \overline{G}_2, V_2^{seen}$).

Input:
- first and second graph to merge $G_1, G_2; \overline{G}_1, \overline{G}_2$
- first and second observers set V_1^{seen}, V_2^{seen}

Initialize:
- $\mathcal{E}' \leftarrow \emptyset$, $\mathcal{G}' \leftarrow (V, \mathcal{E}')$
- $\overline{\mathcal{E}}' \leftarrow \emptyset$, $\overline{\mathcal{G}}' \leftarrow (V, \overline{\mathcal{E}}')$

for $e \in V_1^{seen} \times V_2^{seen}$ **do**
- **if** *(*$e \in G_1$ *and* $e \in \overline{G}_2$*) or (*$e \in G_2$ *and* $e \in \overline{G}_1$*)* **then**
  - **return** $(\emptyset,\emptyset), (\emptyset,\emptyset), conflict$
- **else**
  - **if** $e \in G_1$ *or* $e \in G_2$ **then**
    - $\mathcal{E}' \leftarrow \mathcal{E}' \cup \{e\}$
  - **if** $e \in \overline{G}_1$ *or* $e \in \overline{G}_2$ **then**
    - $\overline{\mathcal{E}}' \leftarrow \overline{\mathcal{E}}' \cup \{e\}$

return $\mathcal{G}', \overline{\mathcal{G}}', success$

Algorithm 4: Replying algorithm.

Input:
the node set V
local node v_l
its sub connectivity graphs $G'(v_l) = (V, E'(v_l))$, $\overline{G}'(v_l) = (V, \overline{E}'(v_l))$
RPC message AskTopo(v_r, $\overline{S_h^r}$)

Initialize:
$\mathcal{E}' \leftarrow E'(v_l)$, $\mathcal{G}' \leftarrow (V, \mathcal{E}')$
$\overline{\mathcal{E}}' \leftarrow \overline{E}'(v_l)$, $\overline{\mathcal{G}}' \leftarrow (V, \overline{\mathcal{E}}')$
$S_h \leftarrow V \setminus \overline{E}'(v_l)$
$(\mathcal{G}_f', \overline{\mathcal{G}_f}', status) \leftarrow FetchAndCombine(V, vl, G'(v_l), \overline{G}'(v_l), S_h \cap \overline{S_h^r}, \emptyset)$

if *status* = *success* **then**
return ReplyTopo(G_f', $\overline{G_f}'$, $\{v_l\} \cup S_h \cap \overline{S_h^r}$)

else
return ReplyTopo($\mathcal{G}'$, $\overline{\mathcal{G}}'$, $\{v_l\}$)

Generally speaking, the workflow is that, the initiator v_i starts the fetching process by calling FetchAndCombine($V, v_i, G'(v_i), \overline{G}'(v_i), S_h(v_i), \overline{S_h}(v_i)$), which will send AskTopo RPCs to all reachable remote nodes. This will trigger the process of Algorithm 4 in all these remote nodes. All the receivers then try to reply to v_i ReplyTopo messages containing their sub connectivity graphs and possibility containing the sub connectivity graphs of nodes cannot be reached by the sender. The function of FetchAndCombine will call the function Merge, which is presented in Algorithm 3, to merge different sub graphs after checking and resolving the conflict. Our default conflict resolving policy is that: if two sides give the detection result of an edge as a Pending and a non-Pending, the resolving result will be the non-Pending result; if two sides give the detection result of an edge as two different non-Pending results, the merging process will stop and give out the result of "conflict".

For example, if v_i and v_j detects the edge (v_i, v_j) to be *Healthy* and *Pending*, respectively, after merging, the edge will be marked as *Healthy* which is to say the edge will appeared in the Healthy edge set E'; if v_i and v_j detects the edge (v_i, v_j) to be *Healthy* and *Unhealthy*, respectively, merging will be interrupted. Table 4 shows the detail of this policy.

Table 4. Edge merging policy.

Edge State 1	Edge State 2	Merge Result
Healthy	*Healthy*	*Healthy*
Unhealthy	*Unhealthy*	*Unhealthy*
Healthy	*Pending*	*Healthy*
Unhealthy	*Pending*	*Unhealthy*
Pending	*Pending*	*Pending*
Healthy	*Unhealthy*	Conflict

4.2.3. Leader Execution Module

Leader node chosen by the former module uses leader execution module to find the set of V_R and removes these nodes in V_R from the cluster. Then, for a leader node v_l and an arbitrary remote node v_x, the downing rule can be described as follows:

$$r_d(v_l, v_x) = \begin{cases} TRUE, & \text{iff } v_x \in V_R, \\ FALSE, & \text{otherwise.} \end{cases} \tag{23}$$

We construct V_R based on the idea that after the execution, the rest of the nodes can normally transmit data with each other, which is to say they form a complete graph from the perspective of graph theory. So we abstract the objective of this module to be (1) find a maximal clique, with best effort, a maximum clique, of G', which means finding a subgraph $K'_p = \left(V_m, \binom{V_m}{2}\right)$ of G' with maximal or maximum nodes satisfying the condition that K'_p is a complete graph; (2) let $V_R := V \setminus V_m$.

The MCP (max clique problem) is a NP-Complete problem [24]. Therefore, we need an algorithm to reduce the computing complexity to make our algorithm available in big clusters. In this section, according to the applicability of transitivity, we propose two algorithms for each case. In the first case with the property, our algorithm will always find a max clique with the computing complexity of $\mathcal{O}(|V|)$. While in the second case without the property, we propose simple heuristic algorithm that will find a maximal (maybe max) clique with the computing complexity of $\mathcal{O}(|V|^2)$.

The first scenario is that the transitivity is applicable. In this case, the leader v_l can find a max clique by a very simple policy of:

$$\begin{aligned} &\text{let } V_m \leftarrow \{v : v \in V \setminus \{v_l\}, (v_l, v) \in E\} \cup \{v_l\}, \\ &K_p = \left(V_m, \binom{V_m}{2}\right), \text{where } p = |V_m|. \end{aligned} \tag{24}$$

This is because the theorem below can be proved.

Theorem 5. *With the transitivity, a healthy node and all its peers end with healthy links construct the unique max clique of the global graph G.*

$$\begin{aligned} &G = (V, E) \text{ is an undirected graph.} \\ &S_m \text{ is a set of max cliques of } G. \\ &\exists v \text{ s.t. } deg_G(v) > [\frac{|V|}{2}] - 1 \overset{(2)}{\Bigg\}} \Rightarrow \begin{cases} (1) |S_m| \equiv 1, \\ \\ (2) \quad \exists K_p = \left(V_m, \binom{V_m}{2}\right), \\ \quad V_m = \{v_x : (v, v_x) \in E\} \cup \{v\} \text{ s.t. } K_p \in S_m. \end{cases} \end{aligned} \tag{25}$$

Proof. We let $E_m = \binom{V_m}{2}$, $V_R = V \setminus V_m$. We also introduce the notations of K_p, S_m, V_m, E_m, G, V, E in Theorem 5 (25).

Maximal proof:

Because the definition of V_R means a set of nodes with unhealthy link state with at least one node in set V_m, which can be expressed as: $\forall v \in V_R : \exists v_h \in V_m \; s.t. \; (v_h, v) \notin E_m$. We can then conclude that $\forall v_f \in V_R :$, the graph $\widetilde{G} = (V_m \cup \{v\}, E_m \cup \{(v_f, v_h) : v_h \in V_m, (v_f, v_h) \in E\})$ is not a complete graph, which means the graph $K_p = (V_m, E_m)$ is a maximal clique of the graph G.

Maximum and unique proof:

We assume that,

$$\exists K_x = \left(V_x, \binom{V_x}{2}\right), K_x \neq K_p, V_x \subset V \text{ s.t. } x \geqslant p, \text{ where } x = |V_x|. \tag{26}$$

We denote $V_p = V_R \cap V_x$, $V_q = V_m \cap V_x$. If (26) is true, and because $p > [\frac{|V|}{2}] - 1$, we can infer that $V_p \neq \emptyset$, $V_q \neq \emptyset$. Then $\exists v_p, v_q, v_p \in V_R, v_q \in V_m$ s.t. $v_p \in K_x, v_q \in K_p$. Thus we have,

$$\exists v_p, v_q, v_p \in V_R, v_q \in V_m \text{ s.t. } (v_p, v_q) \in E. \tag{27}$$

On the other hand, $\forall v_p \in V_R$, $\exists v_q \in V_m$ s.t. $(v_p, v_q) \notin E$. According to Lemma 3, we have the inference that,

$$\forall v_p \in V_R, \forall v_q \in V_m, \nexists (v_p, v_q) \text{ s.t. } (v_p, v_q) \in E. \tag{28}$$

(27) conflicts with (28), thereby the assumption (26) is false. Thus, $\nexists K_x \neq K_p$ s.t. $x \geqslant p$.

□

Therefore, in this case, the leader node v_l can simply construct the global G' from $G'(v_l)$ and $\overline{G}'(v_l)$ to approximate G. With a loose policy that a leader will not remove a node with the link state of Pending between them, the G' can be built with:

$$G' = (V, E), E' = \binom{V}{2} \setminus \overline{E}'(v_l). \tag{29}$$

Finally, with (24) and (29), we find the set V_R:

$$V_R = \{v : (v_l, v) \notin E'\} \tag{30}$$

The second scenario is that the transitivity is not applicable. In this case, we propose an easy-understanding, easy-implementing algorithm that will always find a maximal clique K_p of G. Our idea is that, from the Self-Checking module, we can see that the healthy state of a node v_x has strong correlation with $deg(v_x)$. We therefore iteratively remove the node v_f with worst state, i.e., minimal $deg(v_f)$ until the rest of the nodes are fully connected. Algorithm 5 shows the process of this algorithm in details.

Algorithm 5: Finding a maximal clique.

Input: global graph $G = (V, E)$
Initialize:
  $K_p = (V_p, E_p) \leftarrow G$
  $V_f \leftarrow \emptyset$
while $|E_p| < \frac{|V_p|(|V_p|-1)}{2}$ **do**
  $R_{cand} \leftarrow \{v : v \in V_p, deg_{K_p}(v) = \min_{v_i \in V_p} deg(v_i)\}$
  $v_f \leftarrow$ a random $v \in R_{cand}$
  $V_p \leftarrow V_p \setminus \{v_f\}$
  $E_p \leftarrow E_p \setminus \{(v, v_f) : v \in V_p \setminus \{v_f\}, (v, v_f) \in E_p\}$
  $V_f \leftarrow V_f \cup \{v_f\}$
for $v_f \in V_f$ **do**
  if $\forall v_h \in V_p$ *s.t.* $(v_f, v_h) \in E$ **then**
    $E_p \leftarrow E_p \cup \{(v_f, v) : v \in V_p\}$
    $V_p \leftarrow V_p \cup \{v_f\}$
if $|V_p| < [\frac{|V|}{2}]$ **then**
  Alert and prevent this leader execution.
Output: K_p

After the leader get the clique $K_p = \left(V_m, \binom{V_m}{2}\right)$, it will find the set V_R to be:

$$V_R = \{v : v \notin V_m\} \tag{31}$$

5. Performance Evaluation

5.1. Simulation Setup

5.1.1. Baseline Methods

We compare our evaluation model with four algorithms as follows: (1) Φ accrual failure detector (PFD) [8] is a commonly used adaptive failure detector, which assumes that the interval of heartbeat responses follows normal distribution. Specifically, they define a metric of link state ϕ by $\phi = -\log_{10}(1 - F(timeSinceLastHeartbeat))$, where F is the cumulative distribution function of a normal distribution with mean and standard deviation estimated from historical heartbeat inter-arrival times. By comparing ϕ with a threshold T_ϕ, it gives out the state of a link. (2) Exponential Distribution Failure Detector (EDFD) is an adaptive failure detector [9], which assumes the interval of heartbeat responses follows exponential distribution. Specifically, it defines a metric of link state E_d by $E_d = F(timeSinceLastHeartbeat)$, where $F(t) = 1 - e^{-\frac{1}{\mu}t}$. By comparing E_d with a threshold T_{ed}, it gives out the state of a link. (3) 2WFD is an adaptive failure detector [10] that optimizes the Chen FD [7]. It uses two windows with different sizes, i.e., size of n_1 and n_2, to store the interval of recent heartbeats. By comparing current time T_{now} and the predicted time $\tau_{l+1} = \max(EA_{l+1}^{n_1}, EA_{l+1}^{n_2}) + \alpha$, it gives out the state of a link. In the formula above, $EA_{l+1}^{n_1}$ and $EA_{l+1}^{n_1}$ are the next heartbeat exptected time calculated from the two windows respectively. (4) Calculating the coefficient of variation of network latency is an approach to evaluate the severity of packet loss rate of a link. It quantifies the jitter C by calculating the coefficient of variation of the collected RTTs. The coefficient of variation is calculated by: $C = \frac{\sigma}{\mu}$, where σ stands for the standard deviation, and μ stands for the mean value. By comparing C with a threshold T_c, it gives out the state of a link. We denote this algorithm as CV. Also, we denote our proposed algorithm as AV.

We also compare our system-level testing results with a simulated controller cluster. In the simulated controller cluster, we implement a cluster of nodes which uses the original Downing mechanism, namely AutoDown, which we have already discussed before (Recall Table 2).

5.1.2. Evaluation Metrics

In order to measure the correctness of the link evaluation methods compared with the true link state, we use a well-established and widely-used metric in binary classification to quantify the detection accuracy, i.e., F1-score [25]. Specifically, F1-score is defined based on precision rate and recall rate [25]. Precision rate can be expressed as $P(M_h, N_h) = \frac{\mathcal{M}_h}{\mathcal{N}_h}$. We denote $\mathcal{M}_h$ to be the number of healthy markers on a healthy link while $\mathcal{N}_h$ is the number of evaluations on this link. Then, recall rate can be expressed as $R(M_f, N_f) = \frac{\mathcal{M}_f}{\mathcal{N}_f}$. We denote $\mathcal{M}_f$ to be the number of unhealthy markers on a unhealthy link while $\mathcal{N}_f$ is the number of evaluations on this faulty link. Based on them, F1 score is defined as: $F1 = \frac{2 \times P \times R}{P+R}$. A higher $F1$ score indicates a better performance on an evaluating approach.

To measure the performance of system-level testing results, we focus on the detection rate and mis-kicking rate. The detection rate $\mathcal{R}_D$ indicates the speed to detect and remove faulty nodes in the cluster. Because in our testing environment, once a faulty node is kicked, it will restart immediately and it takes about $T_{setup} = 2$ min to startup. We then define the detection rate to be $\mathcal{R}_D = \frac{\mathcal{N}_{fr} \times T_{setup}}{\mathcal{S}_f \times T_{run}}$, where $\mathcal{S}_f$ denotes how many faulty nodes are there in this test case, N_{fr} denotes how many times faulty nodes being removed and T_{run} denotes the total testing duration of this test. The mis-kicking rate $\overline{R}_k$ indicates the severity of incorrect Downing process which causes healthy nodes being removed from

the cluster. We define the mis-kicking rate to be $\overline{\mathcal{R}_k} = \frac{\mathcal{N}_r - \mathcal{N}_{fr}}{\mathcal{N}_r}$, where N_r denotes the total quantity of downing actions. A higher $\mathcal{R}_D$ and a lower $\overline{\mathcal{R}_k}$ indicate a better performance.

5.1.3. Simulation Scenarios

We run three groups of experiments to check the accuracy of our algorithm compared with baseline methods. In the first group of experiment, we compare the precision-recall and F1 score with fixed optimized parameter and varying packet loss rate. In the second group of experiment, we compare the precision-recall and F1 score with a fixed packet loss rate and varying parameters, i.e., number of records of RTT and thresholds. In the last group, we apply our algorithm on a real-world product and compare the result with the baseline method.

5.2. Evaluation Results

5.2.1. Adaptability of Environment

We present the link evaluation results with varying packet loss rate of the three algorithms in Figure 4. This group of experiment indicates the adaptability of each algorithm with different packet loss rates. We fix the parameters of each algorithm. The parameters of each algorithm are set as follows. For PFD, the size of historical heartbeats is set to 100, and the threshold T_ϕ is set to 0.45. For EDFD, the size of historical heartbeats is set to 1000, and the threshold T_{ed} is set to 0.65. For 2WFD, the size of large window n_1 is set to 2000, the size of small window n_2 is set to 10, and the decision time α is set to 0.

For AV and CV, N_w (number of records) is set to 30. The thresholds for AV are: $T_{safe} = 0.6$, $T_{alert} = 1.5$. The threshold for CV is $T_c = 1.0$. From the results, we can observe that our algorithm achieves the highest F1 score in each network environment, which proves that our evaluation method is accurate in detecting the severity of different packet loss rates.

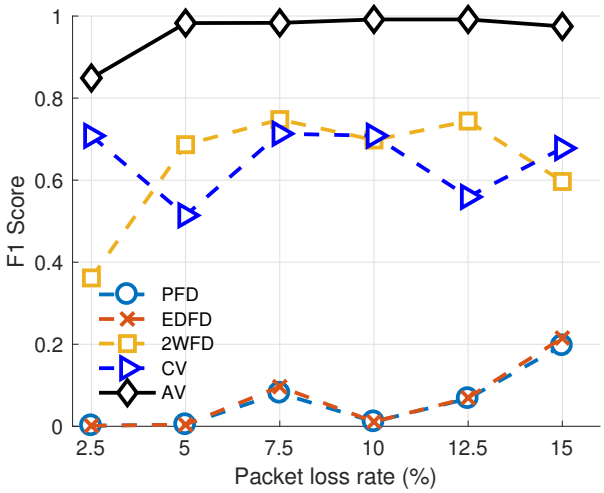

Figure 4. Performance of Evaluation model vs. packet loss rates.

5.2.2. Impact of Parameters

Figure 5 shows the impact of parameters on the performance of each algorithm. In this group of experiment, we fix the packet loss rate to 12.5%. Figure 5a shows the result with varying thresholds. We select 4 groups of typical thresholds for each algorithm. Table 5 shows the thresholds we use. We fix the size of historical heartbeats of PFD, EDFD, 2WFD the same as the latter group of experiment, and fix the length of records N_w of AV and CV to be 30. Figure 5b shows the result with varying record

sizes. We check four different groups of sizes of records and fix the other parameters as same as the former experiment (Adaptability of Environment).

Table 6 shows the count of records we use in this group of experiment.

Results show that, with different parameters, the performance of our algorithm is higher than the other four algorithms. Specifically, the accuracy can be improved by adjusting the number of records N_w to higher values.

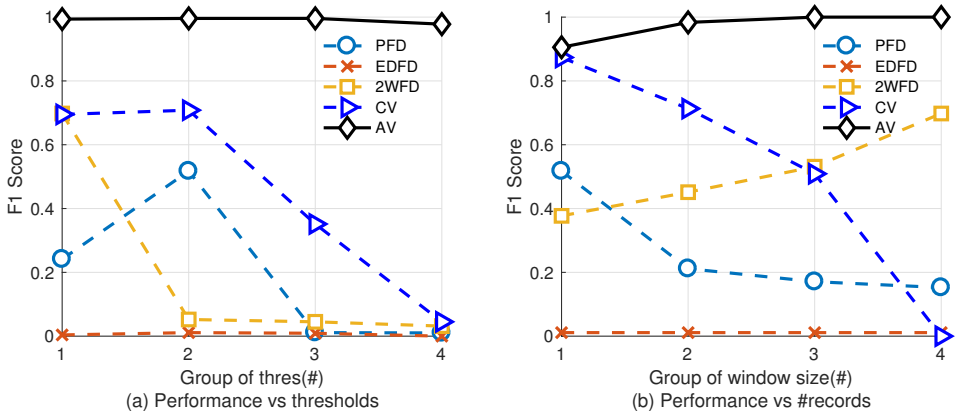

Figure 5. Performance of Evaluation model vs. parameters.

Table 5. Thresholds used by experiment on parameters.

Algorithm	Threshold #1	Threshold #2	Threshold #3	Threshold #4
PFD	$\phi = 0.05$	$\phi = 0.25$	$\phi = 0.45$	$\phi = 1.5$
EDFD	$T_{ed} = 0.6$	$T_{ed} = 0.7$	$T_{ed} = 0.8$	$T_{ed} = 0.9$
2WFD	$\alpha = 0$	$\alpha = 10$ ms	$\alpha = 20$ ms	$\alpha = 30$ ms
CV	$T_c = 0.6$	$T_c = 1.0$	$T_c = 1.5$	$T_c = 2.0$
AV	$T_{safe} = 0.6$ $T_{alert} = 1.5$	$T_{safe} = 1.0$ $T_{alert} = 3.0$	$T_{safe} = 1.5$ $T_{alert} = 6.0$	$T_{safe} = 2.0$ $T_{alert} = 10.0$

Table 6. Window sizes used by experiment on parameters.

Algorithm	Record Size #1	Record Size #2	Record Size #3	Record Size #4
PFD	100	500	1000	2000
EDFD	100	500	1000	2000
2WFD	$n_1 = 100$ $n_2 = 1$	$n_1 = 1000$ $n_2 = 1$	$n_1 = 1000$ $n_2 = 10$	$n_1 = 2000$ $n_2 = 10$
CV	10	30	60	400
AV	10	30	60	400

Results of Real-world Testing To test and verify the reliability of our algorithm, we apply the algorithm on a product named AgileController. This product is provided by Huawei Inc. It is developed based on a famous open source SDN controller, i.e., OpenDaylight. OpenDaylight supports the feature of constructing a SDN controller cluster to provide most of the advantages brought by distributed systems. Specifically, the OpenDaylight project adopts Akka cluster service, which provides a decentralized and gossip protocol-based membership management service. Thus, we implement PingBasedDown algorithm as a plugin of Akka, and hook this plugin into this product. The AgileController has the ability to automatically restart if it is shut down. The startup process takes about 2 min if restarted. We first construct a cluster of AgileControllers containing several nodes in their simulated production environment. Then we injure failures into some of these nodes with various packet loss rate by the tool named TC of linux operating system. During testing, if one node

is removed by a leader, a log containing the information of this removing action, e.g., who is being removed, is generated. After a relatively long period of time, we collect the logging data and calculate $\mathcal{R}_D$ and $\overline{\mathcal{R}_k}$ of each test.

The scenario of our simulated production environment is that, each controller runs in a virtual machine. The hosts of these virtual machines are within the same data center with network bandwidth of 1 Gbps. The average transmission delay is around 400 µs. The physical jitter of the network latency is low but the garbage collection process in JVM causes extra jitter when measuring the latency. Based on this environment, we set the parameters of PingBasedDown algorithm as follows. The length of records N_w is set to be 30, filter strength F_S is set to be 0.15, latency positioning factor L_{pos} is set to be 0.2, and the two thresholds T_{alert}, T_{safe} are set to be 17.0 and 2.0, respectively.

During 8 days, we do 15 groups of testing that cover different numbers of nodes, different packet loss rates, different numbers of faulty nodes. To show the benefit of our proposed algorithm, we also provide 6 groups of results with AutoDown algorithm. Table 7 shows the results of this experiment using PingBasedDown algorithm, while Table 8 shows the results using the original AutoDown mechanism provided by Akka. For example, the 7th row in Table 7 means that, we run this group of test with a cluster with 5 nodes. The 1st node and the 3rd node (ordered by $h(v)$ ascendingly) are injured with packet loss failures, and the packet loss rate is set to 15% and 40%, respectively. The result of this group is that, the detection rate $\mathcal{R}_D$ reaches 61.3% and the mis-kicking rate $\overline{\mathcal{R}_k}$ is 0.0%. Compared with the baseline method, we can see that when the packet loss rate is as low as 15%, the $\mathcal{R}_D$ of AutoDown is less than 9.1%, while that of our algorithm ranges from 44.7% to 54.4%. This indicates that our algorithm can detect nodes with packet loss issues faster than AutoDown. The $\overline{\mathcal{R}_k}$ of AutoDown ranges from 8.9% to 48.1%, which indicates a poor performance on the accuracy of downing action. With our algorithm, the $\overline{\mathcal{R}_k}$ in all test cases reach a perfect low rate 0.0%, which means that each removing action removes only faulty nodes. Therefore, we can conclude that the results prove the effectiveness and robustness of our proposed algorithm.

Table 7. The system-level testing results. Each group is tested for 12 h.

Size of Cluster	Faulty Nodes	Packet Loss Rate (%)	Detection Rate $\mathcal{R}_D$ (%)	Mis-Kicking Rate $\overline{\mathcal{R}_k}$ (%)
5	1	15	44.7	0
5	1	25	58.3	0
5	1	40	61.9	0
5	1, 3	15, 15	47.5	0
5	1, 3	25, 25	58.6	0
5	1, 3	40, 40	60.2	0
5	1, 3	15, 40	61.3	0
5	1, 3	25, 40	60.8	0
5	1, 3	15, 25	57.5	0
7	1, 2, 3	15, 15, 15	54.4	0
7	1, 2, 3	25, 25, 25	56.1	0
7	1, 2, 3	40, 40, 40	66.4	0
7	1, 2, 3	15, 40, 25	63.1	0
7	1, 2, 3	40, 40, 15	67.2	0
7	1, 2, 3	25, 25, 40	58.3	0

Table 8. The system-level testing results with the original AutoDown mechanism of Akka. Each group is tested for 6 h .

Size of Cluster	Faulty Nodes	Packet Loss Rate (%)	Detection Rate $\mathcal{R}_D$ (%)	Mis-Kicking Rate $\overline{\mathcal{R}_k}$ (%)
5	3, 4	15, 15	5.0	29.4
5	3, 4	25, 25	32.3	43.4
5	3, 4	40, 40	31.9	8.9
7	4, 5, 6	15, 15, 15	9.1	27.8
7	4, 5, 6	25, 25, 25	50.2	48.1
7	4, 5, 6	40, 40, 40	57.7	33.0

6. Conclusions

In this work, we propose an algorithm that consists of Evaluation model and Decision model. This algorithm solves the problem of reduced availability in decentralized clusters when nodes occur to randomly drop packets. Driven by the application layer data, the Evaluation model estimates the link and gives relatively accurate evaluation on link quality. With the link evaluations, the Decision model further identifies the only leader. By modeling the cluster to a simple undirected connectivity graph, the leader finds a max clique of this graph. Then, the leader removes the nodes which are not in this clique in which way to make the cluster more stable and available. Classified by the applicability of transitivity, we simplified the NP-Complete maximum clique problem to a linear and a square complexity algorithm. We then evaluate our algorithm with statistical data. Moreover, we implement our solution and adopt it in a real-world product. All these results show that our approach is highly adaptable and available.

Author Contributions: Hangyu Fan designed the PingBasedDown algorithm, implemented the link evaluation model and decision model, collected and conducted analysis of the experimental data and did the comparative experiment. Huandong Wang investigated the related work, implemented different kinds of failure detection algorithms as baseline methods and helped to formulate the system into the mathematical model. Yong Li helped to provide the simulated production environment in Huawei Inc. and to polish the language and expression of the article.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Wikipedia. Computer Cluster—Wikipedia, The Free Encyclopedia. Available online: https://en.wikipedia.org/w/index.php?title=Computer_cluster&oldid=802238893 (accessed on 20 October 2017).
2. Kahani, M.; Beadle, H.W.P. Decentralised approaches for network management. *ACM SIGCOMM Comput. Commun. Rev.* **1997**, *27*, 36–47.
3. Xiong, N.; Yang, Y.; Cao, M.; He, J.; Shu, L. A survey on fault-tolerance in distributed network systems. In Proceedings of the IEEE International Conference on Computational Science and Engineering (CSE'09), Vancouver, BC, Canada, 29–31 August 2009; Volume 2, pp. 1065–1070.
4. Sari, A.; Akkaya, M. Fault tolerance mechanisms in distributed systems. *Int. J. Commun. Netw. Syst. Sci.* **2015**, *8*, 471–482.
5. Renesse, R.V.; Minsky, Y.; Hayden, M. *A Gossip-Style Failure Detection Service*; Springer: London, UK, 1998; pp. 55–70.
6. Bertier, M.; Marin, O.; Sens, P. Implementation and Performance Evaluation of an Adaptable Failure Detector. In Proceedings of the IEEE International Conference on Dependable Systems and Networks (DSN), Washington, DC, USA, 23–26 June 2002; pp. 354–363.
7. Chen, W. On the quality of service of failure detectors. *IEEE Trans. Comput.* **2002**, *51*, 13–32.
8. Hayashibara, N.; Dfago, X.; Yared, R.; Katayama, T. The ϕ Accrual Failure Detector. In Proceedings of the 23rd IEEE International Symposium on Reliable Distributed Systems, Florianpolis, Brazil, 18–20 October 2004; pp. 66–78.

9. Xiong, N.; Vasilakos, A.; Yang, Y.; Wei, S.; Qiao, C.; Wu, J. *General Traffic-Feature Analysis for an Effective Failure Detector in Fault-Tolerant Wired and Wireless Networks*; Technical Report; Georgia State University: USA, 2011.
10. Tomsic, A.; Sens, P.; Garcia, J.; Arantes, L.; Sopena, J. 2W-FD: A failure detector algorithm with QoS. In Proceedings of the Parallel and Distributed Processing Symposium, Hyderabad, India, 25–29 May 2015; pp. 885–893.
11. Liu, J.; Wu, Z.; Wu, J.; Dong, J.; Zhao, Y.; Wen, D. A Weibull distribution accrual failure detector for cloud computing. *PLoS ONE* **2017**, *12*, e0173666.
12. Turchetti, R.C.; Duarte, E.P.; Arantes, L.; Sens, P. A QoS-configurable failure detection service for internet applications. *J. Internet Serv. Appl.* **2016**, *7*, 9.
13. Ganesh, A.J.; Kermarrec, A.M.; Massoulie, L. Peer-to-peer membership management for gossip-based protocols. *IEEE Trans. Comput.* **2003**, *52*, 139–149.
14. Akka. Akka Introduction. Available online: https://doc.akka.io/docs/akka/current/scala/guide/introduction.html (accessed on 20 October 2017).
15. Sommers, J.; Barford, P.; Duffield, N.; Ron, A. A geometric approach to improving active packet loss measurement. *IEEE/ACM Trans. Netw.* **2008**, *16*, 307–320.
16. Wu, H.; Gong, J. *Packet Loss Estimation of TCP Flows Based on the Delayed ACK Mechanism*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 540–543.
17. Basso, S.; Meo, M.; Martin, J.C.D. Strengthening measurements from the edges: Application-level packet loss rate estimation. *ACM* **2013**, *43*, 45–51.
18. Sun, X.; Coyle, E.J. Low-complexity algorithms for event detection in wireless sensor networks. *IEEE J. Sel. Areas Commun.* **2010**, *28*, 1138–1148.
19. Cerulli, R.; Gentili, M.; Raiconi, A. Maximizing lifetime and handling reliability in wireless sensor networks. *Networks* **2014**, *64*, 321–338.
20. Yim, S.J.; Choi, Y.H. An adaptive fault-tolerant event detection scheme for wireless sensor networks. *Sensors* **2010**, *10*, 2332–2347.
21. Şinca, R.; Szász, C. Fault-tolerant digital systems development using triple modular redundancy. *Int. Rev. Appl. Sci. Eng.* **2017**, *8*, 3–7.
22. Aysal, T.C.; Yildiz, M.E.; Sarwate, A.D.; Scaglione, A. Broadcast gossip algorithms for consensus. *IEEE Trans. Signal Process.* **2009**, *57*, 2748–2761.
23. Peng, K.; Lin, R.; Huang, B.; Zou, H.; Yang, F. Node importance of data center network based on contribution matrix of information entropy. *J. Netw.* **2013**, *8*, 1248–1254.
24. Feige, U.; Goldwasser, S.; Lovasz, L.; Saila, S.; Szegedy, M. Approximating clique is almost NP-complete. In Proceedings of the 32nd Annual Symposium on Foundations of Computer Science, San Juan, Puerto Rico, 1–4 October 1991; pp. 2–12.
25. Powers, D.M. Evaluation: From precision, recall and F-measure to ROC, informedness, markedness and correlation. *J. Mach. Learn. Technol.* **2011**, *2*, 37–63.