Smart Grid 智能电网, 2015, 5(5), 242-251
Published Online October 2015 in Hans. http://www.hanspub.org/journal/sg
http://dx.doi.org/10.12677/sg.2015.55029

Hans汉斯

Research on Fast Recovery from Link Failures Based on Openflow in Smart Grid

Yuanfeng Huang[1], Yu Zhao[1], Bingbing Liao[2], Yongbo Wang[2]

[1]Zhuhai Power Supply Bureau of Guangdong Power Grid Corporation, Zhuhai Guangdong
[2]Guangdong YTD Technology Development Co. Ltd., Guangzhou Guangdong
Email: 13902537756@139.com, 13825603652@139.com

Received: Oct. 2nd, 2015; accepted: Oct. 14th, 2015; published: Oct. 21st, 2015

Abstract

In this paper, we design and evaluate algorithms for fast recovery from link failures in a smart grid communication network, addressing all three aspects of link failure recovery: 1) link failure detection, 2) algorithms for computing backup multicast trees, and 3) fast backup tree installation. Firstly, we design link-failure detection and reporting mechanisms that use Openflow to detect link failures when and where they occur inside the network. Openflow is an open source framework that cleanly separates the control and data planes for use in network management and control. Secondly, we formulate a new problem, MULTICAST RECYCLING, which computes backup multicast trees that aim to minimize control plane signaling overhead. We prove that MULTICAST RECYCLING is at least NP-hard and presents a corresponding approximation algorithm. Lastly, two control plane algorithms are proposed that signal data plane switches to install pre-computed backup trees. An optimized version of each installation algorithm is designed that finds a near minimum set of forwarding rules by sharing rules across multicast groups, thereby reducing backup tree installation time and associated control state. We implement these algorithms in the POX Openflow controller and evaluate them using the Mininet emulator, quantifying control plane signaling and installation time.

Keywords

Smart Grid, Failure Recovery, Openflow

文章引用: 黄远丰, 赵煜, 廖兵兵, 王勇波. 基于 Openflow 的智能电网网络链路故障恢复研究[J]. 智能电网, 2015, 5(5): 242-251. http://dx.doi.org/10.12677/sg.2015.55029

基于Openflow的智能电网网络链路故障恢复研究

黄远丰[1]，赵　煜[1]，廖兵兵[2]，王勇波[2]

[1]广东电网有限责任公司珠海供电局，广东 珠海
[2]广东益泰达科技发展有限公司，广东 广州
Email: 13902537756@139.com, 13825603652@139.com

收稿日期：2015年10月2日；录用日期：2015年10月14日；发布日期：2015年10月21日

摘　要

本文设计了一种智能电网中的链路故障快速恢复算法，并对算法进行了验证。本文方法解决了链路故障恢复三个方面的问题：1) 链路故障探测，2) 计算备份多播树的算法，3) 快速备份树的建立。首先，本文设计了基于Openflow的链路故障探测和上报机制，来探测链路故障。Openflow是一种开源框架，将网络管理的控制和转发平面进行分离。其次，本文设计了多播回收机制，通过计算备份多播树来最小化控制信号的开销。本文证明了多播回收问题是一个NP难问题，并提出一种相应的算法来解决该问题。最后，本文设计了一种优化算法，通过在多播组之间共享规则来查找到接近最优的转发集合，同时降低了备份树和控制机制之间的时间开销。本文在POX Openflow控制器上使用Mininet仿真器评估了所提出的算法，量化了控制平面的信号和安装时间。

关键词

智能电网，故障恢复，Openflow

1. 引言

电力通信网包括车载变电站集合，由电力变电站、发电机中心、电力负载汇聚点组成，集合中的组件通过输电线路来连接。电网的操作可以通过高频率电压以及现有的相量测量设备(PMUs)进行优化。

PMU 器件具有严格的超低数据包延迟和丢包的问题。如果不能满足，PMU 器件会丢失一个重要的电力事件，例如雷击，潜在的导致了不正确策略的级联制定，以及相应动作的发生。例如，闭环回路控制应用需要每个数据包 8~16 ms 的请求延迟[1]，如果每个包并没有在规定时间窗内到达，会执行一个错误的操作。最坏情况下，会导致智能电网失效，诸如印度的智能电网失效[2]。

因此，具有敏感和互通要求的网络规定 PMU 数据必须提供严格的端到端数据保证[1]。为此，互联网的最佳服务模式并不适用于 PMU 器件的严格包转发策略和丢包率限制[3]，需要建立新的网络架构或者对现有的网络协议进行优化，文献[1] [3] [4]提出了高效的网内快速转发和链路交换机故障恢复机制。由于 PMUs 通过多个位置将数据分发到器件上[1]，因此多播需要在数据分发时对数据进行表示分发。

软件定义网络实现了交换机和路由器数据转发平面和控制平面的分离，实现控制平面的可编程，将转发规则可编程的写到交换机或路由器上。控制平面和数据平面的互通，包括消息转发规则，较为常用的是 Openflow 协议[5]。

本文使用 Openflow 定义了新的控制平面算法，并对算法进行优化，以实现对重要智能电网数据的分发，算法安装在网络交换机上。本文设计了网络链路故障的快速恢复算法，链路故障后，数据包不能从该链路进行转发。算法从三个方面对网络链路故障进行了恢复，分别为链路故障检测、预先计算备份多播树以及实现快速备份树的安装。

设计链路故障检测和上报算法 Pcount，使用 Openflow [5]协议检测算法故障。网路内部检测可以降低算法发现故障并进行恢复的时间。但现有方法[6]主要针对端到端的包检测，导致了检测速度较慢。

设计一种经典的优化问题来计算备份多播树。本文定义了一个新问题，多播循环，在链路故障发生之前预先计算备份多播树，最小化备份树的安装时间。该问题优化不同于现有方法[7] [8]关注于最大化备份路径和主要路径之间的节点和路径脱节。

证明了多播回收问题是一个 NP 难问题，并提出了一种优化算法，BUNCHY。

提出 MERGER，它是一种在多播模式下的 Openflow 应用，目的是降低转发状态。MERGER 使用本地优化方法来创建一个近似最小集合，该集合通过对具有普通子节点的多播树进行转发规则的合并。

设计了两种算法——PROACTIVE 和 REACTIVE 算法——来快速建立备份树。PROACTIVE 预先建立备份树转发规则，并且在检测到链路故障之后触发这些规则，检测到链路故障后使用 REACTIVE 算法来建立备份树。

提供了一种算法执行原型，APPLESEED，使用 POX 模型，并使用 Mininet 验证了算法。PCOUNT、BUNCHY、MERGER、PROACTIVE 和 REACTIVE 均在 POX [9]里执行，POX 是一种开源 Openflow 控制器，每个算法使用 Mininet 仿真器[10]进行了验证。

本文组织结构如下，第二部分介绍相关工作，第三部分描述 PMU 器件需求，并介绍 Openflow 协议。第四部分描述算法，第五部分对所提算法进行仿真实验，第六部分总结全文。

2. 相关工作

Gridstat [1]是一个出版订阅系统，是首个考虑实现智能电网数据分发系统的研究项目。Gridstat 是构建在现有网络协议(例如 IP，MPLS)上的 overlay 网络，主要关注制定 PMU 器件更加具有鲁棒性要求的网络协议。

现有工作大部分关注基于端到端的测量来检测包丢失[6]，要求多次检测来确定丢包率。本文算法 PCOUNT 提出了一种更快速准确的单一链路包检测方法，直接测量出网路内部的准确流量。

不同于多播的情况，预先计算备份树的创新点是 MPLS 快速重路由算法来重路由时间优先的单播 IP 流[11]。现有工作中计算备份树使用本地优化标准计算路径的方法，例如单一多播树特定条件，本文还考虑了网络标准，例如多个多播树的特定条件。此外，现有工作中并没有对最小化控制面板信号进行研究，本文在多播回环问题中，对该问题进行解决。备份路径或者备份树是通过计算与主路径脱节的最大节点来得出的[7] [12]。

3. 预言

3.1. PMU 器件及其 QoS 需求

本文设计了算法对多播智能电网数据的可靠性，特别是 PMU 器件 QoS 要求的数据进行处理。参考文献[13]详细介绍了 PMU 器件的 QoS 要求细节。本文算法通过提供可靠的数据分发，使得 PMU 器件可以执行准确及时的操作。

3.2. 名词和假设

将通信网络建模为 $G=(V,E)$，这里的 $(u,d)\in E$ 表示一条从 u 到 d 的链路，V 包括三种节点：能够

发送 PMU 数据的节点(PMU 发送节点)，接受 PMU 数据的节点(数据接受节点)，连接 PMU 发送节点和数据接受节点的交换机。我们假设 G 具有 $m \geq 1$ 多播树来发送 PMU 数据。用 $T = \{T_1, T_2, \cdots, T_m\}$ 表示在 G 中的 m 个多播树， $T_i = (V_i, E_i, r, S)$ 表示一个以 r 为根节点的多播树， r 的直接相邻的边为 E_i， 向量 V_i 表示从 r 到每个 $s \in S$ 的链路。用 $\omega(T_i)$ 表示所有 T_i 边的总权重。

用 $T_i^l = (V_i^l, E_i^l, r, s \in S)$ 表示具有 l 条边的第 i 颗树，其中 $l \in E_i^l$ 。对每个定向树 T_i^l 中的每条链路 l，定义备份树 $\hat{T}_i^l = (\hat{V}_i^l, \hat{E}_i^l, r, S)$，备份树是一个有向树，根节点是 r，根节点到每个 $s \in S$ 的链路是有向路径，例如 $l \notin \hat{E}_i^l$ 。T_i^l 和 $\hat{T}_i^l$ 表示原始树和备份树。对每一个原始树 $T_i^l = (V_i^l, E_i^l, r, S)$，具有一个多播流 $f_i = (r, S)$，数据接受节点 $S = \{d_1, d_2, \cdots, d_k\}$， 每个 $d_i \in S$ 具有一个端到端包延迟需求，并且具有一个丢包率需求。

3.3. Openflow

Openflow 是一个开源标准将交换设备的控制平面和转发平面进行了分离，并提供了一个可编程的控制框架[5]。所有 Openflow 算法和协议通过一个逻辑集中的控制器进行控制，同时网路交换设备根据本地转发规则进行包转发。

Openflow 提供给了一个交换机流表，每个交换机使用对每个操作进行匹配的方法来对包进行操作[5]：流表根据数据包的包头进行匹配，判断将该数据流进行转发还是丢弃。交换机对每个流实例保存数据。

Openflow 并没有提供多播转发模式，因此需要设计多播模式，本文称之为 Basic。Basic 是一个多播 IP 地址来标记多播组，并使用这些地址来设定多播树交换机的流表。对每个多播树 $T_i^l = (V_i^l, E_i^l, r, S)$，Basic 在每个交换机 V_i 上安装一个流表。流表通过组的多播地址来匹配包，并将每个包的备份进行转发，相应出端口的交换机出口连接是 E_i 。

4. 问题描述

本文提出了一个算法集合，称之为 APPLESEED，在链路故障时提高了多播树的鲁棒性，运行在一个 Openflow 控制器上，目标是最小化丢包率，保证满足端到端延迟的需求。APPLESEED 由三部分组成：PCOUNT 算法进行快速链路检测，快速安装并计算备份树，快速安装预先计算备份树。

4.1. 使用 Openflow 进行链路故障检测

本文提出了 PCOUNT 算法，使用 Openflow 来检测链路故障。网内检测可以降低丢包发生的时间。快速包检测对 PMU 器件来说是至关重要的，PMU 器件对丢包率具有极高的敏感度。现有工作[6]大多关注于使用端到端的测量来评估丢包率，导致了检测时间超长。PCOUNT 考虑当丢包了超过一定门限时，将该链路定义为失效。本文定义的一个链路是指：两个 Openflow 交换设备之间的端到端连接称之为一条链路。文章[13]中，详细介绍了 PCOUNT 算法检测多个交换机之间的丢包率。

对一个长度为 w 的时间窗口，PCOUNT 评估一个链路 (u, d) 的丢包率，检测在链路 (u, d) 上的流 $M = \{f_1, f_2, \cdots, f_k\}$ 的丢包率，步骤如下：

1) 安装规则来计算所有在节点 d 收到的标签 f_i 的包。对在节点 d 上的每个流 $f_i \in M$ ，安装流表，使用标签模块在第二步的节点 u ，对流打标签。

2) 在节点 u 对所有流 $f_i \in M$ 打标签。假设 u 使用流表 e_i 来匹配和转发流 f_i 的包。首先，PCOUNT 算法生成一个唯一的标签，然后，对每个流 f_i ，PCOUNT 算法创建一个新的流表 e_i' ， e_i' 是流表 e_i 的备份，不同的是，e_i' 包括数据包 dl_vlan 区域的标签。e_i' 比 e_i 具有更高的 Openflow 优先级，来保证每个 f_i 数据流均能被打上标签。

3) 在 w 时间窗口执行完之后，停止对节点 u 打标签，同时对 u, d 进行包统计。每个打标签规则在 u 都是单独进行查询的，在 d 进行一个聚合查询。

4.2. 计算备份树

APPLESEED 预先计算一个备份树，当 PCOUNT 监测到一条链路故障时，预先计算的备份树对这条故障链路进行恢复。本文在这里提出了一个新问题多播回收，来计算备份树并且提出了相应的算法 BUNCHY。

4.2.1. 多播回收问题

多播回收问题的目标是计算备份树来最大化初始树的边。回收初始树使用 SDN 控制器来保证初始树的转发规则安装到网络里，而不是新建立一套转发规则。在检测到链路故障时，备份树开始建立，并恢复之前的网络链路。

初始树 $T_i^l=\left(V_i^l,E_i^l,r,S\right)$，它的备份树是 $\hat{T}_i^l=\left(\hat{V}_i^l,\hat{E}_i^l,r,S\right)$，我们定义一个二进制向量 c_v^l，对所有的 $v\in\hat{V}_i^l$，如果 v 具有相同的前向树 T_i^l 和 $\hat{T}_i^l$，这时 $c_v^l=0$。否则，$c_v^l=1$。对于 T_i^l 和 $\hat{T}_i^l$，定义：

$$c_i^l=\sum_{\forall v\in\hat{V}_i^l}c_v^l \tag{1}$$

c_i^l 是需要安装 $\hat{T}_i^l$ 新规则的数量。

我们将多播回收问题定义为改进的斯坦纳树问题，称之为 STEINER-ARBORESCENCE。对与根节点 r 和 S 来说，用 $SA_i\left(G\right)=\left(V,E,r,S\right)$ 表示在 G 上计算斯坦纳树，根节点是 r，生成树是 S，这里的 $r,S\in T_i$。

多播回收的输入是 $\left(G,T^l,l,\alpha\right)$，这里的 $G=\left(V,E\right)$ 是一个有向图，$T^l=\left\{T_1^l,T_2^l,\cdots,T_k^l\right\}$ 中的每个 $T_i^l\in T^l$ 是使用 $l\in E$ 的初始树，这里 $\alpha\geq1$。对每个初始树使用 $l\in E$ 的输出是一个备份树。备份树的集合可以是 $\hat{T}^l=\left\{\hat{T}_1^l,\hat{T}_2^l,\cdots,\hat{T}_k^l\right\}$：

$$\begin{aligned}&\min\quad&&\sum_{1\leq i\leq k}C_i^l\\&s.t.\quad&&w\left(\hat{T}_i^l\right)\leq\alpha\cdot w\left(SA_i\left(G'\right)\right),\forall\hat{T}_i^l\in\hat{T}^l\end{aligned} \tag{2}$$

这里，$G'=\left(V',E'\right)$，$E'=E-\left\{l\right\}$，$w\left(\hat{T}_i^l\right)$ 表示 $\hat{T}_i^l$ 的节点权重之和。目标函数是最大化重使用的初始树的边，同时 α 的约束是如何在最小化 C_i^l 的同时生成的备份树的最大值。

定理 4.1. 多播回收问题是一个 NP 难问题

证明详见参考文献[13]。

4.2.2. Bunchy 最优化算法

Bunchy 是解决多播回收问题的简单最优化算法。对每个链路 l，APPLESEED 算法使用 BUNCHY 对每个初始树使用链路 l 来计算备份树。

给定 $\left(G,T^l,l,\alpha\right)$，对每个 $T_i^l\in T^l$，Bunchy 通过两个步骤来计算 $\hat{T}_i^l$。首先，生成一个 G 的备份 $G'=\left(V',E'\right)$，并且将 l 从 E' 中删除。设定链路权重 $e\in T_i^l$ 为 0，设定 $e\notin T_i^l$ 为 1。然后，对 G' 运行 STEINERP- ARBORESENCE 优化算法，并且使用 $\hat{T}_i^l$ 作为结果。如果 $\hat{T}_i^l$ 满足公式(2)，将 $\hat{T}_i^l$ 作为最优解，否则，返回错误。

设定初始树的链路权重为 0，在第一步中，STEINERP-ARBORESENCE 优化算法不产生任何成本的使用初始树的链路，促进了初始树边的重使用。如果 Bunchy 算法在第二步骤中返回错误，则需要将 α 的取值调整的更大，或者生成一个新的多播树来满足树大小的限制。

4.3. 安装备份树

安装备份树有两个步骤：首先，计算需要安装转发规则的交换机的数目，以及相应的需要生成的

Openflow 流表的个数。然后，控制器向相应的交换机发送安装信号，来安装转发规则。本文将介绍两种安装算法：PROACTIVE 和 REACTIVE。这两种算法均计算转发规则，来同时生成一个单一的备份树。初始树为 $T_i^l=\left(V_i^l,E_i^l,r,S\right)$，备份树为 $l\in E^l$，$\hat{T}_i^l=\left(\hat{V}_i^l,\hat{E}_i^l,r,S\right)$。

REACTIVE 算法：首先确定需要新转发规则的节点。如果 T_i^l 和 $\hat{T}_i^l$ 在节点 u 使用同一个输出链路，那么在节点 u 不需要安装新的转发规则。转发规则仅在 $v\in\hat{V}^l/V^l$ 和每个 $v\in\hat{V}^l\bigcap V^l$，节点在 T_i^l 和 $\hat{T}_i^l$ 具有不同的输出链路。对每个节点，REACTIVE 算法预先计算流表来匹配 T_i 多播地址的包。最后，当 l 失效时，REACTIVE 触发交换机来安装预先计算的转发规则。

PROACTIVE 算法：该算法在初始链路 l 失效之前，计算并安装备份树流表。由于该算法仅触发单一节点的备份安装，因此故障恢复速度比较快。当由于转发操作错误引起的链路故障，该算法不能不加修改的安装备份。对一个节点使用该算法时，初始树和备份树的节点输出链路必须是不同的。

为了解决该问题，PROACTIVE 算法对每个备份树指派了一个唯一的备份树 id，表示为 bid。每个备份树使用 bid 进行流表匹配和包转发，将 bid 值写入到 dl_src 区域。当备份树 $\hat{T}^l$ 被触发时，$\hat{T}^l$ 的根节点将 bid 的值写入到 dl_src 包头，表示这些包应该由 $\hat{T}^l$ 来转发而不是由 T^l 来进行转发。

4.4. 优化多播树安装

Basic 多播安装需要在每个多播树节点创建一个流表，使用多播树地址来匹配输入包。因此，一个交换机可能有多个流表实例来执行相同的转发操作。作为 Basic 的优化算法，我们提出了 Merger 算法，该算法计算 Openflow 转发规则的近似最小集合，通过在每个节点将流表中具有相同输出链路的规则进行聚合。MERGER 算法降低了故障恢复的备份树安装规模。

5. 实验验证

本文使用 POX 控制器[9]来执行上述第四章节中的算法。POX 控制器基于 Openflow 1.0 并且运行在 Mininet 2.0.0 版本[10]上。实验环境是使用 Linux 操作系统，具有 2.33 GHz Intel(R) Xeon® CPU，15 GB RAM。Mininet 的配置是运行在 Oracle Virtal Box 虚拟机上，并且安装了 4 GB RAM 和单核 CPU。使用 Mininet 软件交换机，Open vSwitch，APPLESEED 运行在 VirtualBox VM 上。

5.1. 链路故障检测器

本文运行两个模拟器来评估 PCOUNT 算法，首先，我们度量 PCOUNT 算法的丢包率，然后评估当 PCOUNT 监测到更多的数据流时交换机和控制器的性能。

丢包率评估的准确度：如图 1 所示 PCOUNT 测量链路 (u,d) 上的丢包率，我们生成 m 个多播组，每个 $h_1,h_2,\cdots,h_m$ 多播包向接受节点 $s_1,s_2,\cdots,s_m$ 以每秒钟 60 个数据包的速率发送数据包。Basic 算法使用多播模式，实现了从节点 u 到 d 的 m 个流表。本文设定 $m=\{10,20,30,40,50\}$，使用 Minniet 模拟器，通过节点 u 到 d 的数据包服从贝努利分布，丢包率为 $p=\{0.01,0.05,0.10\}$。

本文对 PCOUNT 算法丢包率的准确度进行了度量，优化了 PCOUNT 算法的检测器数据流参数，对丢包率实现了更为准确的度量。PCOUNT 对每一个监测到的数据流进行计算，表明如果有失效的数据流发生了错误，该数据流是 PCOUNT 没有监测到的。本文使用一个实例来说明该算法，设定 $m=10$，$p=0.05$。

图 2 对比了 PCOUNT 链路丢包率函数为 $w=\{0.5,1,\cdots,5\}$，置信区间为 95%，结果表明，每个数据流随机选取，监测到 (u,d) 的数据流有 $k=\{10\%,40\%,70\%,100\%\}$，对每个 w,k，置信区间运行 100 次模拟。

PCOUNT 估计丢包率是非常准确的，95%的置信区间，对所有的 w,k，有 15%的丢包率，对于 75 个数据包的情况，80%的模拟运行，预计结果为 5%的丢包率。

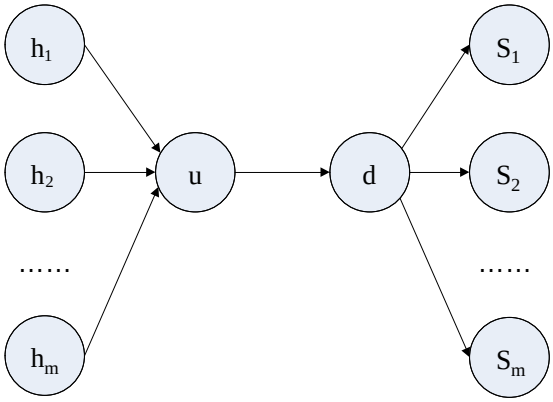

Figure 1. Topology with PCOUNT evaluation
图 1. 使用 PCOUNT 评价的拓扑图

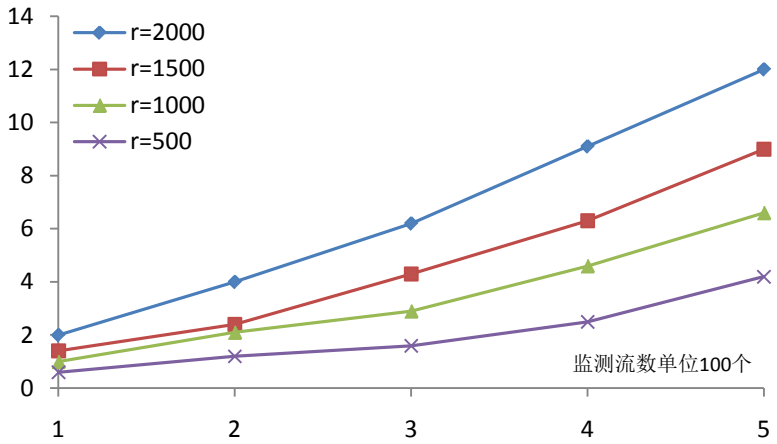

Figure 2. Comparison of execution time, confidence interval is 95%
图 2. 执行时间对比图，置信区间为 95%

执行时间：接下来验证 PCOUNT 算法在监测数据流的规模变大时，执行时间的变化情况。固定时间窗口为 2 秒钟，如图 1 所示，评估 (u,d) 的丢包率。执行时间是指当 PCOUNT 算法发送第一个静态队列到 PCOUNT 算法计算出丢包率的时间。如果 PCOUNT 算法监测到在链路 (u,d) 上集中 k 数据流的包丢失，该算法将向节点 u 发送一个静态队列，并且发送一个集中数据流队列给节点 d 。Mininet 使用 linux 缺省的调度器，实现的是 CPU 多进程，保证了交换机为空时，多线程的处理 PCOUNT 请求。

图 2 表明了算法的总体执行时间作为 PCOUNT 算法的数据流数目的一个输入，k，这里每个数据点的平均计算时间均通过超过 50 次的实验模拟得出。为了评估流表大小对队列执行时间的影响，本文在 u 和 d 上设定了 r 个流表项。算法的执行时间大部分都是在交换机上处理 $k+1$ 个静态队列的时候消耗掉的，执行时间根据 k 成倍数增长，在每个 r 流表项之间都会产生一段较长的时间间隔。本文的实验结果表明，为了获得合理的执行时间，只监控少于 75 次的流就可以满足实验要求。

由于交换机在每次模拟之间是空闲的，因此，CPUs 在软件交换机上所产生的能耗比硬件交换机模式下将更多[14]，本文的实验结果低估了处理时间。此外，PCOUNT 算法执行贝努利试验会丧失部分优越性，原因是在所有流中的丢包率是均匀的，并且丢包的发生也是有规律的。同时，实验结果并没有统计和监测大规模流队列的丢包情况。因此，使用 PCOUNT 算法的实际效果应该比贝努利试验得到的效果更优越。

5.2. 备份树安装模拟器

在本部分，我们将模拟单一链路故障的情况，并且测量故障恢复时间，交换机存储开销，控制平面信号，以及 PROACTIVE 算法和 REACTIVE 算法在没有进行 MERGER 优化情况下的垃圾回收开销。由于篇幅限制，本文仅说明了控制平面的信号结果，并简单介绍了总体实验情况。实验的详细步骤和结果参见文献[13]。

设定：使用 IEEE 总线系统 14, 30, 57 和 118[4]，以及基于 IEEE 总线系统的合成图。合成图的生成方法，参见参考文献[13]，该文献中使用了 IEEE 总线系统作为一个临时模板生成具有相同度分布的合成图。

我们假设物理总线系统拓扑的网络总体沟通成本是可以全部检测到的，一个 Openflow 交换机安装在每个总线上，也安装在两个单向链路之间，每个方向上交换机的配置相同。设定 PMUs 器件测量电压和当前总线的相量，之后将测量结果发送给第一跳链接的交换机，之后通过 Openflow 交换机对 PMU 测量结果进行广播。

对每个总线系统 n 来说，我们生成了 n 个交换机，n 个节点的合成拓扑，并且将链路权重设定为 1。然后，随机生成 $m=\left\{1,2,\cdots,\frac{n}{2}\right\}$ 个多播组，每个组有 $\frac{n}{3}$ 个随机终端节点，使用 STEINER_ARBORESCENCE 近似算法来生成 m 个初始树。BUNCHY 算法中 $\alpha=1.1$，然后对每个初始树进行预先计算，对每个初始树链路生成一个备份树。之后，我们将一个随机链路 l 关闭，制造链路故障，触发 REACTIVE 算法和 PROACTIVE 算法，来设定备份树。对每个 m，生成 35 个不同的有向树，3 个随机的多播组集合。

本文对比了安装备份树的控制信号，在网络中安装备份树的信号是初始树 m 的函数。图 3 表明 REACTIVE 算法运行在 BASIC 和 MERGER 模式下的结果，使用基于 IEEE 总线系统 57 来生成有向拓扑图。实验结果的趋势可以用来说明整个网络的状况。

REACTIVE + LB 计算出链路 l 失效后需要安装的备份树 $\hat{T}^l$ 的最小个数，计算出 $\hat{T}^l$ 在每个节点出向链路的集合数目 v，制定 $\hat{T}^l$ 在 v 转发包的规则。将通过 $\hat{T}^l$ 的节点数相加，并返回其下限值。

信令开销结果：PROACTIVE 比 REACTIVE 的信令开销更小，甚至比 REACTIVE + LB 的开销也小。PROACTIVE 通过使用故障链路给备份树的根节点发送一个控制信息来触发备份树。REACTIVE 必须给多个交换机同时发送信号才能安装备份树。

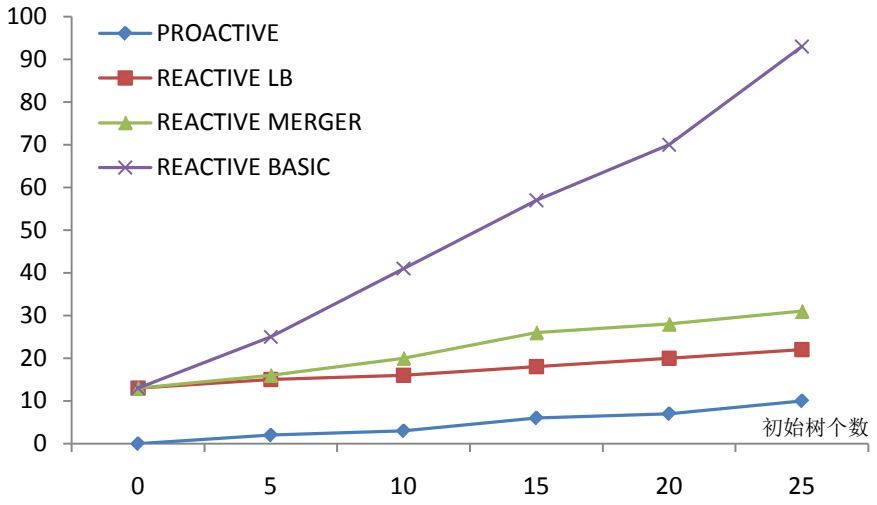

Figure 3. Control signal for triggering backup tree
图 3. 触发备份树的控制信号个数

对 REACTIVE 而言，BASIC 和 MERGER 之间的差距增加了备份树的数目，因为随着 m 的增长，MERGER 可以重复使用备份树的转发规则。实验结果表明当 $m \geq 7$ 时，MERGER 由于重复使用了转发规则，将节省 75%的开销。平均而言，MERGER 仅需要 LB 算法 25%的信令消息，这表明 MERGER 的本地优化并没有因为流的聚合而丧失优越性。

其他实验结果：使用上述相同的实验设定，本文将 m 进行量化，m 是 PROACTIVE 算法根据转发规则需要预先安装的备份树的数目，安装备份树的时间，垃圾回收开销。我们发现这三种评价指标的值与信号开销成正比。

6. 总结

本文对智能电网数据多播的可靠性进行了研究，提出并设计了一种快速丢包监测和快速故障恢复方法，通过预先计算并安装备份树实现了快速的网络链路故障恢复。由于预先计算备份树的方法需要改变现有网络的交换机，因此，本文使用 Openflow 交换机来实现新的转发规则。

首先提出了 PCOUNT 算法，使用 Openflow 协议来预先监测网络内部丢包率。然后，设计多播回收算法来计算备份树，并且重利用多播树的边界规则来降低控制平面的信令。多播回收被证明是 NP 难问题，本文提出了一种近似算法来对该问题进行求解。最后，提出了 PROACTIVE 和 REACTIVE 算法，安装在 Openflow 交换机上来执行。为优化 PROACTIVE 和 REACTIVE 算法，本文提出了 MERGER 算法，将交换机上的转发规则进行合并，这些交换机必须是多播树的普通子节点。

Mininet 模拟器表明当监视短期时间窗的小规模流时，PCOUNT 算法的准确性。通过预先安装备份树，PROACTIVE 算法的时间开销是 REACTIVE 算法的十倍，因为相比 REACTIVE 算法的控制信令发送机制，PROACTIVE 算法必须为每个多播信令交换机安装备份树。最后，我们发现 MERGER 算法比 PREACTIVE 算法降低了 2 倍到 2.5 倍的控制平面信令开销。

参考文献 (References)

[1] Bakken, D., Bose, A., Hauser, C., Whitehead, D. and Zweigle, G. (2011) Smart generation and transmission with coherent, real-time data. *Proceedings of the IEEE*, **99**, 928-951. http://dx.doi.org/10.1109/JPROC.2011.2116110

[2] Yardley, J. and Harris, G. (2012) 2nd day of power failures cripples wide swath of India. http://www.nytimes.com/2012/08/01/world/asia/power-outages-hit-600-million-in-india.html?pagewanted=all&r=1&

[3] Birman, K., Chen, J., Hopkinson, E., et al. (2005) Overcoming communications challenges in software for monitoring and controlling power systems. *Proceedings of the IEEE*, **93**, 1028-1041. http://dx.doi.org/10.1109/JPROC.2005.846339

[4] Bobba, R., Heine, E., Khurana, H. and Yardley, T. (2010) Exploring a tiered architecture for NASPInet. *Innovative Smart Grid Technologies* (*ISGT*), Gaithersburg, 19-21 January 2010, 1-8. http://dx.doi.org/10.1109/isgt.2010.5434730

[5] McKeown, N., Anderson, T., Balakrishnan, H., et al. (2008) Openflow: Enabling innovation in campus networks. *Computer Communication Review*, **38**, 69-74. http://dx.doi.org/10.1145/1355734.1355746

[6] Caceres, R., Duffield, N., Horowitz, J. and Towsley, D. (1999) Multicast-based inference of network-internal loss characteristics. *IEEE Transactions on Information Theory*, **45**, 2462-2480. http://dx.doi.org/10.1109/18.796384

[7] Cui, J., Faloutsos, M. and Gerla, M. (2004) An architecture for scalable, efficient, and fast fault-tolerant multicast provisioning. *IEEE Network*, **18**, 26-34. http://dx.doi.org/10.1109/MNET.2004.1276608

[8] Pointurier, Y. (2002) Link failure recovery for mpls networks with multicasting. Master's Thesis, University of Virginia, Charlottesville.

[9] Mccauley, J. (2010) POX: A python-based Openflow controller. http://www.noxrepo.org/pox/about-pox/

[10] Lantz, B., Heller, B. and McKeown, N. (2010) A network in a laptop: Rapid prototyping for software-defined networks. *Proceedings of the* 9*th ACM SIGCOMM Workshop on Hot Topics in Networks*, Article No. 19. http://dx.doi.org/10.1145/1868447.1868466

[11] Rosen, E., Viswanathan, A., Callon, R., et al. (2001) Multiprotocol label switching architecture. RFC 3031.

[12] Luebben, R., Li, G., Wang, D., Doverspike, R. and Fu, X. (2009) Fast rerouting for IP multicast in managed IPTV networks. 17*th International Workshop on Quality of Service*, Charleston, 13-15 July 2009, 1-5. http://dx.doi.org/10.1109/iwqos.2009.5201406

[13] Gyllstrom, D. (2014) Making networks robust to component failures. Ph.D. Dissertation, University of Massachusetts, Massachusetts.

[14] Rotsos, C., Sarrar, N., Uhlig, S., Sherwood, R. and Moore, A. (2012) OFLOPS: An open framework for openflow switch evaluation. *Proceedings of the* 13*th International Conference on Passive and Active Measurement*, Vienna, 12-14 March 2012, 85-95. http://dx.doi.org/10.1007/978-3-642-28537-0_9