MAKING NETWORKS ROBUST TO COMPONENT FAILURE

A Dissertation Outline Presented

by

DANIEL P. GYLLSTROM

Submitted to the Graduate School of the
University of Massachusetts Amherst in partial fulfillment
of the requirements for the degree of

DOCTOR OF PHILOSOPHY

(Compiled on 01/02/2013 at 14:13)

Computer Science

MAKING NETWORKS ROBUST TO COMPONENT FAILURE

A Dissertation Outline Presented

by

DANIEL P. GYLLSTROM

Approved as to style and content by:

Jim Kurose, Chair

Prashant Shenoy, Member

Deepak Ganesan, Member

Lixin Gao, Member

Lori Clarke, Department Chair
Computer Science

ABSTRACT

MAKING NETWORKS ROBUST TO COMPONENT FAILURE

(COMPILED ON 01/02/2013 AT 14:13)

DANIEL P. GYLLSTROM

B.Sc., TRINITY COLLEGE

M.Sc., UNIVERSITY OF MASSACHUSETTS AMHERST

Ph.D., UNIVERSITY OF MASSACHUSETTS AMHERST

Directed by: Professor Jim Kurose

Communication network components – routers, links connecting routers, and sensors – inevitably fail, causing service outages and a potentially unusable network. Recovering quickly from these failures is vital to both reducing short-term disruption and increasing long-term network survivability. In this thesis, we consider instances of component failure in the Internet and in networked cyber-physical systems, such as the communication network used by the modern electric power grid (termed the *smart grid*). We design algorithms that make these networks more robust to component failure. This thesis divides into three parts: (a) recovery from malicious or misconfigured nodes injecting false information into a distributed system (e.g., the Internet), (b) placing smart grid sensors to provide measurement error detection, and (c) fast recovery from link failures in a smart grid communication network.

First, we consider the problem of malicious or misconfigured nodes that inject and spread incorrect state throughout a distributed system. Such false state can degrade the performance of a distributed system or render it unusable. For example, in the case of network routing algorithms, false state corresponding to a node incorrectly declaring a cost of 0 to all destinations (maliciously or due to misconfiguration) can quickly spread through the network. This causes other nodes to (incorrectly) route via the misconfigured node, resulting in suboptimal routing and network congestion. We propose three algorithms for efficient recovery in such scenarios and evaluate their efficacy.

The last two parts of this thesis consider robustness in the context of the electric power grid. We study a type of sensor, a Phasor Measurement Unit (PMU), currently being deployed in electric power grids worldwide. PMUs provide voltage and current measurements at a sampling rate orders of magnitude higher than the status quo. As a result, PMUs can both drastically improve existing power grid operations and enable an entirely new set of applications, such as the reliable integration of renewable energy resources. However, PMU applications require *correct* (addressed in thesis part 2) and *timely* (covered in thesis part 3) PMU data. Without these guarantees, smart grid operators and applications may make incorrect decisions and take corresponding (incorrect) actions.

The second part of this thesis addresses PMU measurement errors, which have been observed in practice. We formulate a set of PMU placement problems that aim to satisfy two constraints: place PMUs "near" each other to allow for measurement error detection and use the minimal number of PMUs to infer the state of the maximum number of system buses and transmission lines. For each PMU placement problem, we prove it is NP-Complete, propose a simple greedy approximation algorithm, and evaluate our greedy solutions.

Lastly, we design algorithms for fast recovery from link failures in a smart grid communication network. This is a two-part problem: (a) link detection failure and (b) algorithms for pre-computing backup multicast trees. To address (a), we design link-detection failure 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. For part (b), we propose a set of algorithms that precompute backup multicast trees to be used after a link failure. Each algorithm aims to minimize end-to-end packet loss and delay but each uses different optimization criteria to achieve this goal: minimize control overhead, minimize the number of affected flows across all multicast trees, and minimize the number of affected sink nodes across all multicast trees. We implement and evaluate these algorithms in Openflow.

TABLE OF CONTENTS

LIST OF TABLES

Table **Page**

LIST OF FIGURES

Figure **Page**

CHAPTER 3

RECOVERY FROM LINK FAILURES IN A SMART GRID COMMUNICATION NETWORK

3.1 Introduction

An electric power grid consists of a set of buses – electric substations, power generation centers, or aggregation points of electrical loads – and transmission lines connecting those buses. The operation of the power grid can be greatly improved by high frequency voltage and current measurements. Phasor Measurement Units (PMUs) are sensors which provide such measurements. PMUs are currently being deployed in electric power grids worldwide, providing the potential to both (a) drastically improve existing power grid operations and applications and (b) enable an entirely new set of applications, such as real-time visualization of electric power grid dynamics and the reliable integration of renewable energy resources.

PMU applications have stringent and in many cases ultra-low *per-packet* delay and loss requirements. If these per-packet delay requirements are not met, PMU applications can miss a critical power grid event (e.g., lightning strike, power link failure), potentially leading to a cascade of incorrect decisions and corresponding actions. For example, closed-loop control applications require delays of $8 - 16$ ms per-packet [7]. If *any* packet is not received within this time window, the closed-loop control application may take a wrong control action. In the worst case, this can lead to a cascade of power grid failures (e.g., the August 2003 blackout in the USA [1] and the recent power grid failures in India [48]).

[1] `http://en.wikipedia.org/wiki/Northeast_blackout_of_2003`

As a result of this sensitivity, the communication network that disseminates PMU data must provide hard end-to-end data delivery guarantees [7]. For this reason, the Internet's best-effort service model alone is unable to meet the stringent packet delay and loss requirements of PMU applications [11]. Instead, either a new network architecture or enhancements to Internet architecture and protocols are needed [7, 11, 12, 29] to provide efficient, in-network forwarding and fast recovery from link and switch failures. Additionally, multicast should figure prominently in data delivery, since PMUs disseminate data to applications across many locations [7].

In this last piece of our research, we design algorithms for fast recovery from link failures in a Smart Grid communication network. Informally, we consider a link that does not meet its packet delivery requirement (either due to excessive delay or actual packet loss) as failed. Our proposed research divides broadly into two parts:

- **Link detection failure.** Here, we design link-failure detection and reporting mechanisms that use OpenFlow [34] – an open source framework that centralizes network management and control – to detect link failures when and where they occur, *inside* the network. In-network detection is used to reduce the time between when the loss occurs and when it is detected. In contrast, most previous work [5, 14, 25] focuses on measuring end-to-end packet loss, resulting in slower detection times.

- **Algorithms for pre-computing backup multicast trees.** Inspired by MPLS fast-reroute algorithms that are used in practice to quickly reroute time-critical unicast IP flows over pre-computed backup paths [18, 23, 35, 40, 45], we propose a set of algorithms, each of which computes backup multicast trees that are installed after a link failure. We also implement these algorithms in OpenFlow and demonstrate their performance.

Each algorithm computes backup multicast trees that aim to minimize end-to-end packet loss and delay, but each algorithm uses different optimization criteria in achieving this goal: minimizing control overhead (Min-Control), minimizing the number of affected flows across all multicast trees (Min-Flows), and minimizing the number of affected sink nodes across all multicast trees (Min-Sinks). These optimization criteria differ from those proposed in the literature. For example, most previous work [18, 23, 35, 40, 45] uses optimization criteria specified over a *single* multicast tree, while we must consider criteria specified across *multiple* multicast trees. Finally, because the smart grid network is many orders of magnitudes smaller than the Internet [2] and multicast group membership is mostly static in the Smart Grid, we can for the most part avoid the scalability issues of Internet-based solutions [18, 23, 35, 40, 45].

The remainder of this chapter is structured as follows. In the following section 3.2, we briefly survey relevant literature. We outline proposed research in Section 3.3: section 3.3.3 details our research thus far on link-failure detection in OpenFlow, and in section 3.3.5, we outline our algorithms for computing backup multicast trees. Our treatment here is necessarily brief, but we indicate work completed thus far as well as proposed future work. Section 3.4 concludes this chapter with a summary of our proposed research and timeline for future work.

3.2 Background and Related Work

3.2.1 PMU Applications and Their QoS Requirements

The QoS requirements of several PMU applications planned to be deployed on power grids worldwide are presented in Table 3.1, based on [7, 17]. We refer the reader to the actual documents for a description of each PMU application. The end-

[2]For example, it is estimated that fewer than 10^4 routers/switches are needed for a smart grid network spanning the *entire* USA, whereas there are about 10^8 routers in the Internet [7].

PMU Application	E2E Delay	Rate (Hz)	NASPI Class
♡ Oscillation Detection	$0.25 - 3$ secs	$10 - 50$	N/A
♡ Frequency Instability	$0.25 - 0.5$ secs	$1 - 50$	N/A
♡ Voltage Instability	$1 - 2$ secs	$1 - 50$	N/A
♡ Line Temp. Monitoring	5 minutes	1	N/A
△ Closed-Loop Control	$8 - 16$ ms	$120 - 720+$	A
△ Direct State Measurement	$5 - 1000+$ ms	$1 - 720+$	B
△ Operator Displays	1000+ ms	$1 - 120$	D
△ Distributed Wide Area Control	$1 - 240$ ms	$1 - 240$	B
△ System Protection	$5 - 50$ ms	$120 - 720+$	A
△ Anti-Islanding	$5 - 50$ ms	$30 - 720+$	A
△ Post Event Analysis	1000+ ms	< 1	E

Table 3.1. PMU applications and their QoS requirements. The ♡ refers to reference [17] and △ to [7].

to-end (E2E) delay requirement is at the *per-packet* level, as advocated by Bakken et al. [7].

NASPI defines five service classes (A-E) for Smart Grid traffic, each designating qualitative requirements for latency, availability, accuracy, time alignment, message rate, and path redundancy [7]. At one end of the spectrum, service class A applications have the most stringent requirements, while service Class E designates applications with the least demanding requirements.

In this work, we focus on PMU applications with the most stringent E2E delay requirements, such as closed-loop control and system protection. In particular, we create a binary classification of data plane traffic: traffic belonging to critical PMU applications and all other traffic.

3.2.2 OpenFlow

OpenFlow is an open source framework that cleanly separates the control and data planes, and provides a programmable (and possibly centralized) control framework [34]. All OpenFlow algorithms and protocols are managed by a (logically) centralized controller, while network switches/routers (as their only task) forward packets ac-

cording to the flow tables installed by the controller. By allowing a more centralized network control and management framework, the OpenFlow architecture avoids the high storage, computation, and management overhead that plague many distributed network approaches. Our multicast tree repair algorithms benefit from these OpenFlow features (Section 3.3.5).

OpenFlow exposes the flow tables of its switches, allowing the controller to add, remove, and delete flow entries, which determine how switches forward, copy, or drop packets associated with a controller-managed flow. Phrased differently, OpenFlow switches follow a "match and action" paradigm [34], in which each switch *matches* an incoming packet to a flow table table entry and then takes some *action* (e.g., forwards, drops, or copies the packet). Each switch also maintains per-flow statistics (e.g., packet counter, number of bytes received, time the flow was installed) that can can be queried by the controller. In summary, OpenFlow provides a flexible framework for *in-network* packet loss detection as demonstrated by our detection algorithms (Section 3.3.3).

3.2.3 Related Work

3.2.3.1 Smart Grid Communication

The Gridstat project [3], started in 1999, was one of the first research projects to consider smart grid communication. Our work has benefited from their detailed requirements specification [7].

Gridstat proposes a publish-subscribe architecture for PMU data dissemination. By design, subscription criteria are simple to enable fast forwarding of PMU data (and as a measure towards meeting the low latency requirements of PMU applications). Gridstat separates their system into a data plane and a management plane. The management plane keeps track of subscriptions, monitors the quality of service provided

[3] `http://gridstat.net/`

by the data plane, and computes paths from subscribers to publishers. To increase reliability, each Gridstat publisher sends data over multiple paths to each subscriber. Each of these paths is a part of a different (edge-disjoint) multicast tree. Meanwhile, the data plane simply forwards data according to the paths and subscription criteria maintained by the management plane.

Although Gridstat has similarities with our work, their project lacks details. For example, no protocol is provided defining communication between the management and data plane. Additionally, there is no explicit indication if the multicast trees are source-based.

In North America, all PMU deployments are overseen by the North American SynchroPhasor Initiative (NASPI) [12]. NASPI has proposed and started (as of December 2012) to build the communication network used to deliver PMU data, called NASPInet. The interested reader can consult [12] for more details.

Hopkinson et al [29] propose a Smart Grid communication architecture that handles heterogeneous traffic: traffic with strict timing requirements (e.g., protection systems), periodic traffic with greater tolerance for delay, and aperiodic traffic. They advocate a multi-tier data dissemination architecture: use a technology such as MPLS to make hard bandwidth reservations for critical applications, use Gridstat to handle predictable traffic with less strict delivery requirements, and finally use Astrolab (which uses a gossip protocol) to manage aperiodic traffic sent over the remaining available bandwidth. They advocate hard bandwidth reservations – modeled as a multi-commodity flow problem – for critical Smart Grid applications.

3.2.3.2 Detecting Packet Loss

Most previous work [5, 14, 25] focuses on measuring and detecting packet loss on an end-to-end packet basis. Because PMU applications have small per-packet delay requirements (Section 3.2.1), the time delay between when the loss occurs and

when it is detected needs to be small. For this reason, we will investigate detecting lossy links *inside* the network. Additionally, most previous work takes an *active measurement* approach towards detecting lossy links in which probe messages are injected to estimate packet loss. Injecting packets can potentially skew measurements – especially since accurate packet loss estimates require a high sampling probing rate – leading to inaccurate results [9].

Friedl et al. [25] propose a *passive* measurement algorithm that directly measures actual network traffic to determine application-level packet loss rates. Unfortunately, their approach can only measure packet loss after a flow is expired. This makes their algorithm unsuitable for our purposes because PMU application flows are long lasting (running continuously for days, weeks, and even years). For this reason we propose a new algorithm, FAILED-LINK, that provides in-network packet loss detection for long running active flows (Section 3.3.3).

existing

We note that existing Internet routing algorithms (e.g., OSPF, ISIS, BGP) perform in-network detection of link failure, but not of individual packet loss. They do so by having routers exchange "keep-alive" or "hello" messages and detect a link failure when these messages or their acknowledgments are lost.

instead

3.2.3.3 Multicast Tree Recovery

Approaches to multicast fault recovery can be broadly divided into two groups: on-demand and preplanned. In keeping with their best-effort philosophy, most Internet-based algorithms compute recovery paths on-demand [18]. Because the Smart Grid is a critical system and its applications have strict delivery requirements, we focus on preplanned recovery path computations.

To date, nearly all preplanned approaches [18, 23, 35, 40, 45] are implemented (or suggest an implementation) using virtual circuit packet switching, namely, MPLS. For convenience, in the remainder of this section we assume MPLS is the virtual

circuit packet switching technology used, realizing that other such technologies could be used in its place.

Cui et [18] define four categories for preplanned multicast path recovery: (a) link protection, (b) path protection, (c) dual-tree protection, and (d) redundant tree protection. With link protection, a backup path is precomputed for each link, connecting the link's end-nodes [40, 45]. For each destination, a path protection algorithm computes a vertex-disjoint path with the original multicast tree path between the source and destination [45]. The dual-tree approach precomputes a backup tree for each multicast tree. The backup (dual) tree is not required to be node- or link-disjoint with the primary tree but this is desirable [23]. Finally, a redundant tree is node (link) disjoint from the primary tree, thereby ensuring that any destination remains reachable, either by the primary or redundant tree, if any vertex (edge) is eliminated [35].

Distributed multicast recovery schemes, like the ones discussed above, must navigate an inherent trade-off between high overhead and fast recovery (i.e., the time between when the failure is detected and when the multicast tree is repaired is small) [18]. Here, overhead refers to (i) per-router state that needs to be stored and managed and (ii) message complexity. Because preplanned MPLS-based approaches are tailored to support Internet-based applications – typically having a large number of multicast groups and dynamic group membership – scalability is key. Some of the preplanned approaches (dual-tree and redundant trees) focus more on scalability [18, 23, 35], while others (link and path protection schemes) optimize for fast recovery [40, 45].

For two reasons, our algorithms avoid these scalability issues: we use a centralized control architecture (OpenFlow) rather than a distributed one and the Smart Grid operating environment is very different from the Internet-based applications discussed in the literature. Using OpenFlow's centralized architecture, we store all precomputed

backup/recovery paths at the controller. Thus, we avoid the storage and maintenance issues that distributed approaches must address [18, 23, 40, 45]. In other words, OpenFlow allows our algorithms to bypass the inherent trade-off of high overhead and fast recovery, allowing (for the first time) both fast *and* scalable recovery algorithms.

In the case where the controller is unable to manage router and backup path state, we can simply provision more servers to store precomputed paths. Such a situation is unlikely, considering: the encouraging scalability results reported using a centralized Ethane controller [15] (Ethane is a precursor to OpenFlow that also separates the control and data planes), the Smart Grid is many orders of magnitude smaller that the Internet, and multicast group membership is mostly static in the Smart Grid [7].

Finally, our recovery algorithms use different optimization criteria from previous work considered in this document [18, 23, 35, 40, 45]. Each of these earlier approaches uses optimization criteria specified over a *single* multicast tree, while we must consider criteria specified across *multiple* multicast trees. In addition, none of these approaches explicitly optimize for the criteria we consider: control overhead, number of effected flows, number of affected downstream nodes, and the expected disruption caused by future link failures. Instead, previous work computes backup paths or trees that optimize one of the following criteria: maximize node (link) disjointedness with the primary path [18, 23, 35], minimize bandwidth usage [45], or minimize the number of group members which become disconnected (using either the primary or backup path) after a link failure [40].

3.3 Proposed Research

In this section, we present an example problem scenario (Section 3.3.1), which we reference to explain: our link failure detection algorithm (Section 3.3.3), steps to uninstall trees that become disconnected after a link failure (Section 3.3.4), steps

to install backup multicast trees (Section 3.3.4), and our algorithms for computing backup multicast trees (Section 3.3.5).

3.3.1 Example, Problem Scenario, and Basic Notation

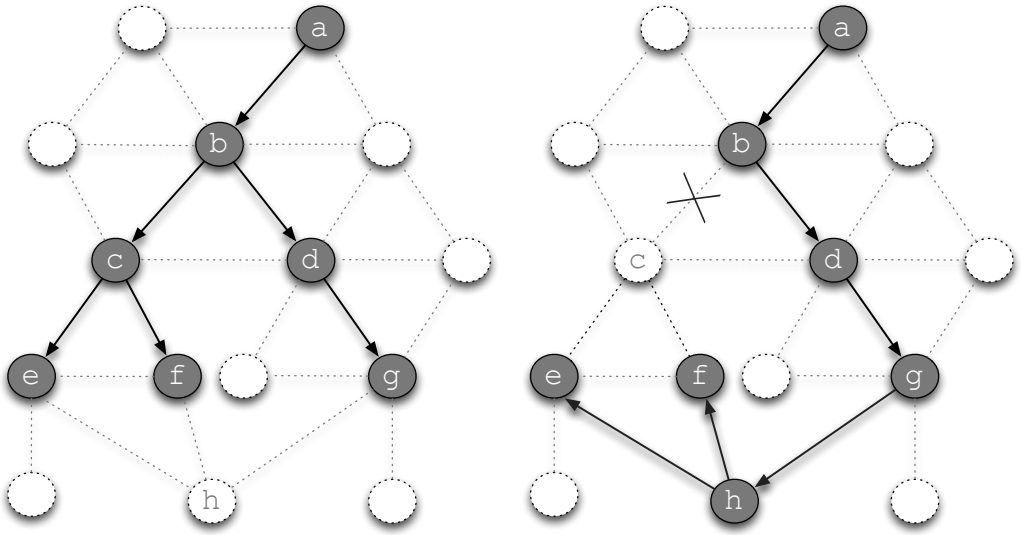
3.3.1.1 Example Scenario

Figure 3.1 depicts a scenario where a single link, (b, c), in a multicast tree fails. Figure 3.1(a) shows a multicast tree rooted at a with leaf nodes (i.e., data sinks) $\{e, f, g\}$. a sends PMU data at a fixed rate and each data sink specifies a per-packet delay requirement. The multicast tree in Figure 3.1(a) uses link (b, c), which we assume fails between the time the snapshots in Figure 3.1(a) and Figure 3.1(b) are taken. When (b, c) fails, it prevents e and f from receiving any packets until the multicast tree is repaired, leaving e and f's per-packet delay requirements unsatisfied. Figure 3.1(b) shows a backup multicast tree installed after (b, c) fails. Notice that the backup tree does not contain any paths using the failed link, (b, c), and has a path between the root (a) and each data sink $(\{e, f, g\})$. In the coming sections we present algorithms that allow multicast trees, like the one shown in Figure 3.1(a), to recover from link failures by installing backup multicast trees, similar to the one in Figure 3.1(a).

such as

3.3.1.2 General Problem Scenario and Basic Notation

Before presenting our algorithms, we first provide a more general problem scenario than the one in Figure 3.1 and introduce some basic notation. We consider a network of nodes modeled as an undirected graph $G = (V, E)$. There are three types of nodes: nodes that send PMU data (PMU nodes), nodes that receive PMU data (data sinks), and switches connecting PMU nodes and data sinks (typically via other switches). We assume G has $m > 1$ source-based multicast trees to disseminate PMU data. Let $T = \{T_1, T_2, \ldots T_m\}$, such that each $T_i = (V_i, E_i) \in T$ is a source-based multicast tree (MT). We assume G only contains MTs in T.



(a) Before link (b, c) fails. (b) After link (b, c) fails.

Figure 3.1. Example used in Section **??**. The shaded nodes are members of the source-based multicast tree rooted at a. The lightly shaded nodes are not a part of the multicast tree.

Each PMU node is the source of its own MT and each data sink has a per-packet delay requirement, specified as the maximum tolerable per-packet delay. *Packet delay* between a sender, s, and receiver, r, is the time it takes s to send a packet to r. We consider any packet received beyond the maximum delay threshold as lost. Note that a data sink's per-packet delay requirement is an end-to-end requirement. Before any link fails, we assume that all PMU data is correctly delivered such that each data sink's per-packet delay requirement is satisfied.

Data sent from a single sender and single data sink is called a *unicast flow*. A unicast flow is uniquely defined by a four-tuple of source address, source port, destination address, and destination port. We assume that data from a single PMU maps to a single port at the source and, likewise, a unique port at the destination.

Because we use multicast, *multicast flows* (as opposed to unicast flows) are used inside the network. Informally, a multicast flow contains multiple unicast flows and

only sends a single packet across a link. We use notation $s, \{d_1, d_2, ...d_k\}$ to refer to a multicast flow containing k unicast flows with source, s, and data sinks $d_1, d_2, ...d_k$. The unicast flows are between s and each $d_1, d_2, ...d_k$. Each multicast flow, $f = s, \{d_1, d_2, ...d_k\}$, has k end-to-end per-packet delay requirements, one for each of f's data sinks. Let F be the set of all multicast flows in G.

We define multicast flows inductively using $T_i \in T$. Starting at the source/root, s, T_i has a multicast flow for each of s's outgoing links $(s, x) \in T_i$. For link (s, x) between s and its one-hop neighbor x, the multicast flow contains all T_i unicast flows that traverse (s, x). For example, the Figure 3.1(a) multicast tree has a single multicast flow $(a, \{e, f, g\})$ at a. Only a single multicast flow is needed because a has only one outgoing link in its multicast tree.

At internal T_i nodes, additional multicast flows are instantiated when T_i branches. Internal node $v \in T_i$, instantiates a new multicast flow for each of v's outgoing links (except for the link connecting v with its parent node in T_i). For outgoing link $(v, u) \in T_i$, v's newly instantiated multicast flow represents T_i's unicast flows that traverse (v, u). In Figure 3.1(a), the $a, \{e, f, g\}$ flow splits when the tree branches at b into multicast flows $a, \{e, f\}$ and $a, \{g\}$. Likewise, multicast flow $a, \{e, f\}$ splits into multicast flows $a, \{e\}$ and $a, \{g\}$ at c.

In keeping with its role as a general framework providing necessary services for programmable networks, OpenFlow does not explicitly provide an implementation for multicast and multicast flows. Instead, OpenFlow switches support a *group table* abstraction that is principally used to implement multicast [39]. Akin to flow table entries for unicast flows, group tables consist of group entries. Each group entry matches incoming packets based on a packet's group ID field – packets are expected to have a unique group ID value in its header – and when appropriate (i.e., when the multicast tree branches at the switch being considered) clones the packet to be sent

out along each of the switch's outgoing links in the multicast tree. [4] This how the concept of "splitting" a flow (introduced in the previous paragraph) is implemented.

One way to implement multicast in OpenFlow is to leverage existing IP multicast protocols as detailed by Kotani et al. [31]. In this approach, the controller assigns a unique group ID to each multicast tree and creates a group table entry, that uses the group ID, at each switch along the multicast tree. Meanwhile, the sender and its first-hop switch use IGMP to set up and manage the controller-generated group IDs. Finally, the sender embeds the group ID in each multicast packet's destination field, allowing for each switch in the multicast tree to identify and forward multicast packets appropriately.

We consider the case where multiple links fail over the lifetime of the network but assume that only a *single link fails at-a-time*. We call the current lossy or faulty link, ℓ. When ℓ fails, all sink nodes corresponding to any multicast flow that traverses ℓ no longer receive packets from the source. As a result, the per-packet delay requirements of each these data sinks is not met. We refer to these data sinks, multicast flows, and the MT associated with each such flow as *directly affected*. In Figure 3.1, $a, \{e, f\}$ is directly affected by (b, c) failing, along with data sinks e and f.

3.3.2 Overview of Our Recovery Solutions and Section Outline

We propose an algorithm, APPLESEED, that is run at the OpenFlow controller to make multicast trees robust to link failures by monitoring and detecting failed links, precomputing backup multicast trees, and installing backup multicast trees after a link failure. [5] As input, APPLESEED is given an undirected graph containing OpenFlow switches; the set of all multicast trees (T); the set of all active multicast

[4] In addition to the group ID, a group entry can optionally match packets using patterns defined over other packet header fields (in the same way we saw with flow entries) such as the source address.

[5] The name APPLESEED is inspired by Johnny Appleseed, the famous American pioneer and conservationist known for planting apple nurseries and caring for its trees.

flows (F); the length of each sampling window, w, used to monitor links and specified in units of time; and, for each multicast flow, a packet loss condition for each link the flow traverses. For now, we restrict packet loss conditions to be threshold-based that indicate the maximum number of packets that can be lost over w time units. The output of APPLESEED is a new set of precomputed backup multicast trees installed in the network and a set of uninstalled multicast trees. All $T_i \in T$ that use the failed link are uninstalled and we call each such T_i a *failed multicast tree*.

We define a *backup multicast tree* for link ℓ and $T_i \in T$ as a multicast tree that has a path between the source, $s \in T_i$, and each data sink $d \in T_i$ that: (a) connects s and d but does not traverse ℓ and (b) satisfies d's per-packet delay requirements. We refer to any multicast tree that satisfies these conditions for ℓ a *backup multicast tree for* ℓ or backup MT for short. In Figure 3.1(b), notice that the installed backup tree has paths $a \rightarrow b \rightarrow d \rightarrow g \rightarrow h \rightarrow \{e, f\}$ connecting a with $\{e, f, g\}$ after (b, c) fails.

APPLESEED divides into three parts:

1. Monitor to detect link failure (e.g., (b, c) in Figure 3.1).

2. Uninstall all trees using the failed link (i.e., failed trees) and install a precomputed backup multicast tree for each uninstalled tree. For each data sink that was disconnected from the root because of the link failure, the backup tree should use a path that routes around the failed link. Note that the newly installed tree will likely require changes to all switches upstream from ℓ, in addition to those at the upstream and downstream ends of link ℓ. Recall in the Figure 3.1 example, data sinks $\{e, f\}$ are reconnected with a in the installed backup tree.

3. Part (2) triggers the computation of a new backup tree for each (backup) tree installed in (2).

APPLESEED uses an OpenFlow-based subroutine (FAILED-LINK) for part (1) and is presented in Section 3.3.3. For part (2), we briefly describe APPLESEED's steps to uninstall and install multicast trees in Section 3.3.4. In Section 3.3.5, we address part (3) by proposing a set of algorithms that compute backup multicast trees.

3.3.3 Link Failure Detection using OpenFlow

In this section, we propose a simple algorithm (FAILED-LINK), used by APPLESEED, that monitors links *inside* the network to detect any packet loss. To help explain FAILED-LINK, we use the example scenario from Section 3.3.1 and refer to a generic multicast tree with an upstream node, u, and downstream node, d.

FAILED-LINK is run at the OpenFlow controller and provides accurate packet loss measurements that are the basis for identifying lossy links. Informally, a lossy link is one that fails to meet the packet loss conditions specified by the controller. We refer to such a link as *failed*. Although APPLESEED is ultimately concerned with meeting the per-packet *delay* requirements of PMU applications, we use packet loss (as opposed to delay) as an indicator for a failed link because OpenFlow provides no native support for timers.

FAILED-LINK has the same input as APPLESEED, specified in Section 3.3.2. The output of FAILED-LINK is any link that has lost packets not meeting the packet loss condition of any multicast flow traversing the link. In the remainder of this document, we assume all flows are multicast and just use *flow* to refer to a multicast flow, unless otherwise specified.

Recall from Section 3.2.2 that each OpenFlow switch maintains a flow table, where each entry contains a match rule (i.e., an expression defined over the packet header fields used to match incoming packets) and action (e.g., "send packet out port 787"). For each packet that arrives at an OpenFlow switch, it is first matched to a flow entry, e, based on the packet's header fields; then e's packet counter is incremented;

and, lastly, e's action is executed on the packet. [6] Note that for our purposes all flow entries are for multicast flows. FAILED-LINK uses these packet counter values to compute per-flow packet loss between two switches over w time units.

FAILED-LINK uses the subroutine, PCOUNT, to measure the packet loss between an upstream node (u) and downstream node (d). PCOUNT does so at the per-flow granularity over a specified sampling window, w, where w is the length of time packets are counted at u and d. For each window of length w, PCOUNT computes packet loss for a flow f, that traverses u and d, using the following steps:

1. **At u, tags and counts all f packets**. We assume, before any changes are made, u uses flow entry e to match and forward f packets. PCOUNT creates a new flow entry, e', that is an exact copy of e, except that e' embeds a version number (i.e., the tag) in the packet's VLAN field. [7] e' is installed with a higher priority than e. In OpenFlow, each flow entry has a corresponding priority specified upon its installation. Incoming packets are matched against flow entries in priority order, with the first matching entry being used. In this way, setting a higher priority for e' than e, ensures that u tags all f packets when e' is installed.

2. **Counts all tagged f packets received at d.** PCOUNT does so by installing a new flow entry at d that matches packets based on the VLAN tag applied at u.

3. **After w time units, turns tagging off at u.** To do so, PCOUNT simply switches the priority of e' and e at u.

---

[6] Not all switches necessarily use OpenFlow. In fact, we anticipate that in practice many switches will not support OpenFlow. For ease of presentation, this section assumes all switches are OpenFlow-enabled.

[7] Note that the VLAN id is one of many fields from packets that can be used to match against flow entries.

4. **Queries u and d for packet counts.** Specifically, the controller uses the OpenFlow protocol retrieve u and d's packet count values for packets tagged by e'. To ensure that all in-transit packets are considered, PCOUNT waits "long enough" (e.g., time proportional to the average per-packet delay between u and d) for in-transit packets to reach d, before reading d's packet counter.

5. **Computes packet loss.** The controller computes packet loss by simply subtracting d's packet count from u's.

In practice, PCOUNT executes step (2) before step (1) to ensure that u and d consider the same set of packets.

In the Figure 3.1 example, the controller uses FAILED-LINK to measure the packet loss for the $a, \{e, f\}$ flow. We assume for link (b, c) and the $a, \{e, f\}$ flow, FAILED-LINK is given a maximum packet loss threshold of 10 packets over w time units. For each sampling window of w time units, PCOUNT instructs b to tag and count all packets corresponding to $a, \{e, f\}$. At the same time, c is instructed by PCOUNT to count the $a, \{e, f\}$ packets tagged by b. Then, the controller uses the OpenFlow protocol to query the packet counter values for $a, \{e, f\}$ at b and c. When (b, c) fails, the packet counter at c for $a, \{e, f\}$ no longer increments, causing a violation of $a, \{e, f\}$'s packet loss threshold for (b, c).

As specified, FAILED-LINK measures packet loss for each *multicast flow* at each link. These flow-level measurements may obfuscate aggregate link-level packet loss. For this reason, we plan to extend FAILED-LINK to group flows together to enable *aggregate* packet loss measurements. Because OpenFlow provides native support for grouping flows and maintains packet counters for each group, FAILED-LINK can be easily extended to group and count packets for any subset of multicast flows traversing the same switch.

PCOUNT's approach for ensuring consistent reads of packet counters bears strong resemblance to the idea of *per-packet consistency* introduced by Reitblatt et al. [41].

Per-packet consistency ensures that when a network of switches change from an old policy to a new one, that each packet is guaranteed to be handled exclusively by one policy, rather than some combination of the two policies. In our case, we use per-packet consistency to ensure that when PCOUNT reads u and d's packet counters, exactly the same set of packets are considered, excluding, of course, packets that are dropped at u or dropped along the path from u to d.

3.3.4 Uninstalling Failed Trees and Installing Backup Trees

After FAILED-LINK detects ℓ's failure, APPLESEED uninstalls all MTs that contain ℓ (i.e., failed trees) and installs a precomputed backup multicast tree for each of these uninstalled tree. Installing backup trees for ℓ, denoted T_ℓ, triggers the computation of new backup trees. A backup MT is needed for each $T_j \in T_\ell$ and for each of T_j's links. Here we only explain how MTs are uninstalled and installed and describe some of side effects of doing so. We postpone explaining h computed until Section 3.3.5.

is the order important? i.e., do you have to install top down or bottom up in the tree?

Uninstalling or installing a multicast tree, T_i, is simple with OpenFlow. The controller sends an instruction to each switch in T_i to remove (add) the flow entry corresponding to T_i. Increased traffic caused by newly installed backup trees may introduce packet loss to multicast flows that do not traverse ℓ but do use a path shared by at least one switch in a backup MT. We refer to these flows as *indirectly affected*. Additionally, we refer to any packet or data sink corresponding to an indirectly affected flow as indirectly affected. In our Figure 3.1 example, $a, \{g\}$ is indirectly affected when the backup MT is installed because the backup MT uses paths $a \rightarrow b \rightarrow d \rightarrow g \rightarrow h \rightarrow \{e, f\}$.

3.3.5 Computing Backup Multicast Trees

Here we present a set of algorithms that compute backup MTs. APPLESEED uses these algorithms in two scenarios. First, as a part of system initialization where a set

of backup MTs are computed for each network link, l; APPLESEED computes a single backup MT for each MT that would be directly affected by l's failure.

Second, APPLESEED triggers the execution of backup tree computations after backup trees, T_ℓ, are installed in response to the most recent link failure, ℓ. APPLESEED reruns the entire computation (as opposed to only computing backup MTs for the newly installed T_ℓ trees): for each network link, l, APPLESEED computes a backup MT for each would-be directly affected MT if l failed. APPLESEED recomputes *all* backup MTs because installing T_ℓ likely changes the network state in terms of the link- or node-disjointedness (or lack thereof) across all system MTs. Therefore, "currently" precomputed backup MTs risk becoming stale since the algorithms used to compute them considered an "old" network state in their optimization. *We hypothesize that recomputing* all *backup MTs each time a single link fails yields a more survivable data dissemination network than only recomputing backup MTs for newly installed backup MTs installed in response to the most recent link failure.*

APPLESEED uses one of the following backup tree computation algorithms: MIN-FLOWS, MIN-SINKS, or MIN-CONTROL. Next, we present initial sketches of each of these algorithms.

this criteria needs to be stated more carefully. I thought the idea was to minimze the number of affected flows. Why is that the same as having the same number of flows cross the link. I don't understand

3.3.5.1 Min-Flows Algorit

First, we outline the MIN-F

MIN-FLOWS ALGORITHM

- Input: (G, T, F, l), such that l is a link in G and each $f \in F$ specifies the maximum per-packet delay it can tolerate.
- Output: A backup multicast tree for each MT that would be directly affected if l were to fail. This set of backup MTs ensure that across *all network links*, the same number of flows traverse each link.

The motivation for targeting a uniform distribution of flows across all network links is to reduce the impact (i.e., number of directly affected flows) of any future link failure.

However, since MIN-FLOWS does not directly consider control overhead, MIN-FLOWS may have high control overhead. This may increase the overall time required to install all backup MTs after a link failure. Investigating this hypothesis is future work.

why does this minimize the number of affected sinks????? I don't see how your optimzation criteria in the OutPut statement match what you are trying to do

3.3.5.2 Min-Sinks Algori

Next, we present the MIN IN-SINKS is closely related to but different from MIN-FLOWS. B minimize the disruption of future link failures but MIN-SINKS considers the expected number of affected sink nodes (as opposed to the expected number of affected flows with MIN-FLOWS).

MIN-SINKS ALGORITHM

- Input: (G, T, F, l), such that l is a link in G and each $f \in F$ specifies the maximum per-packet delay it can tolerate.
- Output: A backup multicast tree for each MT that would be directly affected if l were to fail. These backup MTs ensure that when considering the end-to-end paths used by all network flows, the *set of all end-to-end paths* are distributed such that each network switch has the same number of downstream data sinks.

3.3.5.3 Min-Control Algorithm

Lastly, we outline the MIN-CONTROL algorithm. In effort to provide fast recovery, MIN-CONTROL minimizes the control overhead required to install backup MTs. We assume the control overhead is a function of the number of switches that need to change state (i.e., modify flow tables to uninstall failed MTs or install a backup MT) because the controller must send control signals to each of these switches.

MIN-CONTROL ALGORITHM

is this the same as minimize the sum of the number of routers who's new configuraiton needs to be signalled?

$', T, F, l)$, such that l is a link in G and each $f \in F$ specifies the maximum per-packet delay it can tolerate.

- Output: A backup multicast tree for each MT that would be directly affected if l were to fail. This set of backup MTs minimize the *total* control overhead required by their installation.

Note that MIN-CONTROL does not explicitly consider the control overhead of uninstalling failed MTs because this cost is bounded by the number of failed MTs, which is fixed for each link failure. Intuitively, minimizing control overhead ensures that backup trees are quickly installed. This leads to a fast recovery and ultimately reduces packet loss for all directly affected flows.

3.4 Chapter Conclusion and Future Work

In this final technical chapter, we considered the problem of recovery from link failures in a smart grid communication network. We presented the outline for an algorithm that uses OpenFlow to detect packet loss inside the network of switches and a set of algorithms to precompute backup multicast trees to be installed after a link fails. Our algorithms for computing backup multicast trees differ from those in the literature because each algorithm optimizes for the long-term survivability of the communication network.

For each algorithm proposed in the chapter (APPLESEED, FAILED-LINK, PCOUNT, MIN-FLOWS, MIN-SINKS, MIN-CONTROL), we plan to supplement our basic algorithm description with a detailed specification of the algorithm steps, implement the algorithm in OpenFlow, and evaluate its implementation.

Time permitting, we plan to propose algorithms for computing backup multicast trees that consider indirectly affected flows. Recall from Section 3.3.4 that indirectly affected flows are ones using a path that shares at least one link contained in a

backup tree. As a result, indirectly affected flows may experience packet loss due to increased traffic from redirected flows. We conjecture that an algorithm that minimizes the number of indirectly affected flows reduces the system-wide affects (spatially) of installing backup multicast trees. A similar algorithm is possible that minimizes the number of indirectly affected data sinks.

CHAPTER 4

PLANNED FUTURE WORK

For each algorithm proposed in Chapter 3 – Appleseed, Failed-Link, Pcount, Min-Flows, Min-Sinks, and Min-Control – we plan to supplement its basic description with a detailed specification of the algorithm steps, implement the algorithm in OpenFlow, and evaluate its implementation. We estimate this requires six months of steady work to complete. The author believes this six month goal will be met without issue as the birth and care of a newborn child – the author expects his first born child in January 2013 – should introduce no delays nor complications.

I'd like to see this broken down more, with a timeline and milestones along the way. That will help you (and me). Also, if you aren't going to be doing anything to chapters 1 and 2 then you need to say that explicity (not clear about chapter 2 in particular since you said there is future work there)

BIBLIOGRAPHY

[1] GT-ITM. `http://www.cc.gatech.edu/projects/gtitm/`.

[2] Northeast blackout of 2003. `http://en.wikipedia.org/wiki/Northeast_blackout_of_2003`.

[3] Rocketfuel. `http://www.cs.washington.edu/research/networking/rocketfuel/maps/weights/weights-dist.tar.gz`.

[4] Aazami, A., and Stilp, M.D. Approximation Algorithms and Hardness for Domination with Propagation. *CoRR abs/0710.2139* (2007).

[5] Almes, G., Kalidindi, S., and Zekauskas, M. A one-way packet loss metric for ippm. Tech. rep., RFC 2680, September, 1999.

[6] Ammann, P., Jajodia, S., and Liu, Peng. Recovery from Malicious Transactions. *IEEE Trans. on Knowl. and Data Eng. 14*, 5 (2002), 1167–1185.

[7] Bakken, D.E., Bose, A., Hauser, C.H., Whitehead, D.E., and Zweigle, G.C. Smart generation and transmission with coherent, real-time data. *Proceedings of the IEEE 99*, 6 (2011), 928–951.

[8] Baldwin, T.L., Mili, L., Boisen, M.B., Jr., and Adapa, R. Power System Observability with Minimal Phasor Measurement Placement. *Power Systems, IEEE Transactions on 8*, 2 (May 1993), 707 –715.

[9] Barford, P., and Sommers, J. Comparing probe-and router-based packet-loss measurement. *Internet Computing, IEEE 8*, 5 (2004), 50–56.

[10] Bertsekas, D., and Gallager, R. *Data Networks*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1987.

[11] Birman, K.P., Chen, J., Hopkinson, E.M., Thomas, R.J., Thorp, J.S., Van Renesse, R., and Vogels, W. Overcoming communications challenges in software for monitoring and controlling power systems. *Proceedings of the IEEE 93*, 5 (2005), 1028–1041.

[12] Bobba, R., Heine, E., Khurana, H., and Yardley, T. Exploring a tiered architecture for NASPInet. In *Innovative Smart Grid Technologies (ISGT), 2010* (2010), IEEE, pp. 1–8.

[13] Brueni, D. J., and Heath, L. S. The PMU Placement Problem. *SIAM Journal on Discrete Mathematics 19*, 3 (2005), 744–761.

[14] Cáceres, R., Duffield, N.G., Horowitz, J., and Towsley, D.F. Multicast-based inference of network-internal loss characteristics. *Information Theory, IEEE Transactions on 45*, 7 (1999), 2462–2480.

[15] Casado, M., Freedman, M.J., Pettit, J., Luo, J., McKeown, N., and Shenker, S. Ethane: Taking control of the enterprise. In *ACM SIGCOMM Computer Communication Review* (2007), vol. 37, ACM, pp. 1–12.

[16] Chen, J., and Abur, A. Placement of PMUs to Enable Bad Data Detection in State Estimation. *Power Systems, IEEE Transactions on 21*, 4 (2006), 1608 –1615.

[17] Chenine, M., Zhu, K., and Nordstrom, L. Survey on priorities and communication requirements for pmu-based applications in the nordic region. In *PowerTech, 2009 IEEE Bucharest* (2009), IEEE, pp. 1–8.

[18] Cui, J.H., Faloutsos, M., and Gerla, M. An architecture for scalable, efficient, and fast fault-tolerant multicast provisioning. *Network, IEEE 18*, 2 (2004), 26–34.

[19] De La Ree, J., Centeno, V., Thorp, J.S., and Phadke, A.G. Synchronized Phasor Measurement Applications in Power Systems. *Smart Grid, IEEE Transactions on 1*, 1 (2010), 20 –27.

[20] Dijkstra, E., and Scholten, C. Termination Detection for Diffusing Computations. *Information Processing Letters*, 11 (1980).

[21] El-Arini, K., and Killourhy, K. Bayesian Detection of Router Configuration Anomalies. In *MineNet '05: Proceedings of the 2005 ACM SIGCOMM workshop on Mining network data* (New York, NY, USA, 2005), ACM, pp. 221–222.

[22] Feamster, N., and Balakrishnan, H. Detecting BGP Configuration Faults with Static Analysis. In *2nd Symp. on Networked Systems Design and Implementation (NSDI)* (Boston, MA, May 2005).

[23] Fei, A., Cui, J., Gerla, M., and Cavendish, D. A "dual-tree" scheme for fault-tolerant multicast. In *Communications, 2001. ICC 2001. IEEE International Conference on* (2001), vol. 3, IEEE, pp. 690–694.

[24] Feldmann, A., and Rexford, J. IP Network C0onfiguration for Intradomain Traffic Engineering. *IEEE Network Magazine 15* (2001), 46–57.

[25] Friedl, A., Ubik, S., Kapravelos, A., Polychronakis, M., and Markatos, E. Realistic passive packet loss measurement for high-speed networks. *Traffic Monitoring and Analysis* (2009), 1–7.

[26] Gyllstrom, D., Vasudevan, S., Kurose, J., and Miklau, G. Efficient recovery from false state in distributed routing algorithms. In *Networking* (2010), pp. 198–212.

[27] Gyllstrom, D., Vasudevan, S., Kurose, J., and Miklau, G. Recovery from False State in Distributed Routing Algorithms. Tech. Rep. UM-CS-2010-017, University of Massachusetts Amherst, 2010.

[28] Haynes, T. W., Hedetniemi, S. M., Hedetniemi, S. T., and Henning, M. A. Domination in Graphs Applied to Electric Power Networks. *SIAM J. Discret. Math. 15* (April 2002), 519–529.

[29] Hopkinson, K., Roberts, G., Wang, X., and Thorp, J. Quality-of-service considerations in utility communication networks. *Power Delivery, IEEE Transactions on 24*, 3 (2009), 1465–1474.

[30] Jefferson, D. Virtual Time. *ACM Trans. Program. Lang. Syst. 7*, 3 (1985), 404–425.

[31] Kotani, D., Suzuki, K., and Shimonishi, H. A design and implementation of openflow controller handling ip multicast with fast tree switching. In *Applications and the Internet (SAINT), 2012 IEEE/IPSJ 12th International Symposium on* (2012), IEEE, pp. 60–67.

[32] Lichtenstein, D. Planar Formulae and Their Uses. *SIAM J. Comput. 11*, 2 (1982), 329–343.

[33] Liu, P., Ammann, P., and Jajodia, S. Rewriting Histories: Recovering from Malicious Transactions. *Distributed and Parallel Databases 8*, 1 (2000), 7–40.

[34] McKeown, Nick, Anderson, Tom, Balakrishnan, Hari, Parulkar, Guru M., Peterson, Larry L., Rexford, Jennifer, Shenker, Scott, and Turner, Jonathan S. Openflow: enabling innovation in campus networks. *Computer Communication Review 38*, 2 (2008), 69–74.

[35] Médard, M., Finn, S.G., Barry, R.A., and Gallager, R.G. Redundant trees for preplanned recovery in arbitrary vertex-redundant or edge-redundant graphs. *IEEE/ACM Transactions on Networking (TON) 7*, 5 (1999), 641–652.

[36] Mili, L., Baldwin, T., and Adapa, R. Phasor Measurement Placement for Voltage Stability Analysis of Power Systems. In *Decision and Control, 1990., Proceedings of the 29th IEEE Conference on* (Dec. 1990), pp. 3033 –3038 vol.6.

[37] Mohan, C., Haderle, D., Lindsay, B., Pirahesh, H., and Schwarz, P. ARIES: A Transaction Recovery Method Supporting Fine-Granularity Locking and Partial Rollbacks Using Write-Ahead Logging. *ACM Trans. Database Syst. 17*, 1 (1992), 94–162.

[38] Neumnann, R. Internet routing black hole. *The Risks Digest: Forum on Risks to the Public in Computers and Related Systems 19*, 12 (May 1997).

[39] Pfaff, B., et al. Openflow switch specification version 1.1.0 implemented (wire protocol 0x02), 2011.

[40] Pointurier, Y. Link failure recovery for mpls networks with multicasting. Master's thesis, University of Virginia, 2002.

[41] Reitblatt, M., Foster, N., Rexford, J., and Walker, D. Consistent updates for software-defined networks: Change you can believe in! In *Proceedings of the 10th ACM Workshop on Hot Topics in Networks* (2011), ACM, p. 7.

[42] School, K., and Westhoff, D. Context Aware Detection of Selfish Nodes in DSR based Ad-hoc Networks. In *Proc. of IEEE GLOBECOM* (2002), pp. 178–182.

[43] Vanfretti, L. *Phasor Measurement-Based State-Estimation of Electrical Power Systems and Linearized Analysis of Power System Network Oscillations*. PhD thesis, Rensselaer Polytechnic Institute, December 2009.

[44] Vanfretti, L., Chow, J. H., Sarawgi, S., and Fardanesh, B. (B.). A Phasor-Data-Based State Estimator Incorporating Phase Bias Correction. *Power Systems, IEEE Transactions on 26*, 1 (Feb 2011), 111–119.

[45] Wu, C.S., Lee, S.W., and Hou, Y.T. Backup vp preplanning strategies for survivable multicast atm networks. In *Communications, 1997. ICC 97 Montreal,'Towards the Knowledge Millennium'. 1997 IEEE International Conference on* (1997), vol. 1, IEEE, pp. 267–271.

[46] Xu, B., and Abur, A. Observability Analysis and Measurement Placement for Systems with PMUs. In *Proceedings of 2004 IEEE PES Conference and Exposition, vol.2* (2004), pp. 943–946.

[47] Xu, B., and Abur, A. Optimal Placement of Phasor Measurement Units for State Estimation. Tech. Rep. PSERC Publication 05-58, October 2005.

[48] Yardley, J., and Harris, G. 2nd day of power failures cripples wide swath of india, July 31, 2012. `http://www.nytimes.com/2012/08/01/world/asia/power-outages-hit-600-million-in-india.html?pagewanted=all&_r=1&`.

[49] Zhang, J., Welch, G., and Bishop, G. Observability and Estimation Uncertainty Analysis for PMU Placement Alternatives. In *North American Power Symposium (NAPS), 2010* (2010), pp. 1 –8.