MAKING NETWORKS ROBUST TO COMPONENT FAILURE

A Dissertation 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

May 2013

Computer Science

MAKING NETWORKS ROBUST TO COMPONENT FAILURE

A Dissertation Presented

by

DANIEL P. GYLLSTROM

Approved as to style and content by:

ABSTRACT

MAKING NETWORKS ROBUST TO COMPONENT FAILURE

MAY 2013

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, sensor networks, and 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 repaired 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 centralized network management and control. For part (b), we propose a set of algorithms that repair all multicast trees affected by 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

LIST OF FIGURES

INTRODUCTION

0.1 Component Failure in Communication Networks

0.2 Thesis Overview

0.3 Thesis Contributions

Section Outline

1. 1

2. 2

CHAPTER 1

RECOVERY FROM FALSE ROUTING STATE IN DISTRIBUTED ROUTING ALGORITHMS

1.1 Introduction

Malicious and misconfigured nodes can degrade the performance of a distributed system by injecting incorrect state information. Such false state can then be further propagated through the system either directly in its original form or indirectly, e.g., by diffusing computations initially using this false state. In this paper, we consider the problem of removing such false state from a distributed system.

In order to make the false-state-removal problem concrete, we investigate distance vector routing as an instance of this problem. Distance vector forms the basis for many routing algorithms widely used in the Internet (e.g., BGP, a path-vector algorithm) and in multi-hop wireless networks (e.g., AODV, diffusion routing). However, distance vector is vulnerable to compromised nodes that can potentially flood a network with false routing information, resulting in erroneous least cost paths, packet loss, and congestion. Such scenarios have occurred in practice. For example, in 1997 a significant portion of Internet traffic was routed through a single misconfigured router, rendering a large part of the Internet inoperable for several hours [20]. Distance vector currently has no mechanism to recover from such scenarios. Instead, human operators are left to manually reconfigure routers. It is in this context that we propose and evaluate automated solutions for recovery.

In this paper, we design, develop, and evaluate three different approaches for correctly recovering from the injection of false routing state (e.g., a compromised node incorrectly claiming a distance of 0 to all destinations). Such false state, in turn, may propagate to other routers through the normal execution of distance vector routing, making this a network-wide problem. Recovery is correct if the routing tables in all nodes have converged to a global state in which all nodes have removed each compromised node as a destination, and no node has a least cost path to any destination that routes through a compromised node.

Specifically, we develop three novel distributed recovery algorithms: `2`$^{\text{nd}}$ `best`, `purge`, and `cpr`. `2`$^{\text{nd}}$ `best` performs localized state invalidation, followed by network-wide recovery. Nodes directly adjacent to a compromised node locally select alternate paths that avoid the compromised node; the traditional distributed distance vector algorithm is then executed to remove remaining false state using these new distance vectors. The `purge` algorithm performs global false state invalidation by using diffusing computations to invalidate distance vector entries (network-wide) that routed through a compromised node. As in `2`$^{\text{nd}}$ `best`, traditional distance vector routing is then used to recompute distance vectors. `cpr` uses snapshots of each routing table (taken and stored locally at each router) and a rollback mechanism to implement recovery. Although our solutions are tailored to distance vector routing, we believe they represent approaches that are applicable to other instances of this problem.

For each algorithm, we prove correctness, dervive communication complexity bounds, and evaluate its efficiency in terms of message overhead and convergence time via simulation. Our simulations show that when considering topologies in which link costs remain fixed, `cpr` outperforms both `purge` and `2`$^{\text{nd}}$ `best` (at the cost of checkpoint memory). This is because `cpr` can efficiently remove all false state by simply rolling back to a checkpoint immediately preceding the injection of false routing state. In scenarios where link costs can change, `purge` outperforms `cpr` and `2`$^{\text{nd}}$ `best`. `cpr` performs poorly because, following rollback, it must process the valid link cost changes that occurred since the false routing state was injected; `2`$^{\text{nd}}$ `best` and `purge`, however, can make use of computations subsequent to the injection of false routing state that did not depend on the false routing state. We will see, however, that `2nd best` performance suffers because of the so-called count-to-∞ problem.

Recovery from false routing state has similarities to the problem of recovering from malicious transactions [2, 17] in distributed databases. Our problem is also similar to that of rollback in optimistic parallel simulation [15]. However, we are unaware of any existing solutions to the problem of recovering from false routing state. A related problem to the one considered in this paper is that of discovering misconfigured nodes. In Section 1.2, we discuss existing solutions to this problem. In fact, the output of these algorithms serve as input to the recovery algorithms proposed in this paper.

This chapter has six sections. In Section 1.2 we define the problem and state our assumptions. We present our three recovery algorithms in Section 1.3. Then, in Section 1.4, we briefly state the results of our complexity analysis. Section 1.5 describes our simulation study. We detail related work in Section 1.6 and finally we conclude and comment on directions for future work in Section 1.7.

1.2 Problem Formulation

We consider distance vector routing [4] over arbitrary network topologies. We model a network as an undirected graph, $G = (V, E)$, with a link weight function $w : E \rightarrow \mathbb{N}$. Each node, v, maintains the following state as part of distance vector: a vector of all adjacent nodes ($adj(v)$), a vector of least cost distances to all nodes in G ($\overrightarrow{min_v}$), and a *distance matrix* that contains distances to every node in the network via each adjacent node ($dmatrix_v$).

We assume that the identity of the compromise node is provided by a different algorithm, and thus do not consider this problem in this paper. Examples of such algorithms include [10, 11, 12] in the context of wired networks and [21] in the wireless setting. Specifically, we assume that at time t, this algorithm is used to notify all neighbors of the compromised node(s). Let t' be the time the node was compromised.

For each of our algorithms, the goal is for all nodes to recover "correctly": all nodes should remove the compromised node as a destination and find new least cost distances that do not use the compromised node. If the network becomes disconnected as a result of removing the compromised node, all nodes need only compute new least cost distances to all other nodes within their connected component.

For simplicity, let $\overline{v}$ denote the compromised node, let $\overrightarrow{old}$ refer to $\overrightarrow{min_{\overline{v}}}$ before $\overline{v}$ was compromised, and let $\overrightarrow{bad}$ denote $\overrightarrow{min_{\overline{v}}}$ after $\overline{v}$ has been compromised. **TODO** *check if any notation can be removed.*

1.3 Recovery Algorithms

In this section we propose three new recovery algorithms: `2`$^{\text{nd}}$ `best`, `purge`, and `cpr`. With one exception, the input and output of each algorithm is the same. [1]

Input: Undirected graph, $G = (V, E)$, with weight function $w : E \rightarrow \mathbb{N}$. $\forall v \in V$, $\overrightarrow{min}$ and $dmatrix$ are computed (using distance vector). Also, each $v \in adj(\overline{v})$ is notified that $\overline{v}$ was compromised.

Output: Undirected graph, $G' = (V', E')$, where $V' = V - \{\overline{v}\}$, $E' = E - \{(\bar{v}, v_i) \mid v_i \in adj(\overline{v})\}$, and link weight function $w : E \rightarrow \mathbb{N}$. $\overrightarrow{min_v}$ and $dmatrix_v$ are computed via the algorithms discussed below $\forall v \in V'$.

First we describe a preprocessing procedure common to all three recovery algorithms. Then we describe each recovery algorithm.

[1] Additionally, as input `cpr` requires that each $v \in adj(\overline{v})$ is notified of the time, t', in which $\overline{v}$ was compromised.

1.3.1 Preprocessing

All three recovery algorithms share a common preprocessing procedure. The procedure removes $\overline{v}$ as a destination and finds the node IDs in each connected component. This could be implemented (as we have done here) using diffusing computations [9] initiated at each $v \in adj(\overline{v})$. In our case, each diffusing computation message contains a vector of node IDs. When a node receives a diffusing computation message, the node adds its ID to the vector and removes $\overline{v}$ as a destination. At the end of the diffusing computation, each $v \in adj(\overline{v})$ has a vector that includes all nodes in v's connected component. Finally, each $v \in adj(\overline{v})$ broadcasts the vector of node IDs to all nodes in their connected component. In the case where removing $\overline{v}$ partitions the network, each node will only compute shortest paths to nodes in the vector.

1.3.2 The 2nd best Algorithm

2nd `best` invalidates state locally and then uses distance vector to implement network-wide recovery. Following the preprocessing described in Section 1.3.1, each neighbor of the compromised node locally invalidates state by selecting the least cost pre-existing alternate path that does not use the compromised node as the first hop. The resulting distance vectors trigger the execution of traditional distance vector to remove the remaining false state.

2nd `best` is simple and makes no synchronization assumptions. However, 2nd `best` is vulnerable to the count-to-∞ problem. Because each node only has local information, the new shortest paths may continue to use $\overline{v}$. We will see in our simulation study that the count-to-∞ problem can incur significant message and time costs.

1.3.3 The purge Algorithm

`purge` globally invalidates all false state using a diffusing computation and then uses distance vector to compute new distance values that avoid all invalidated paths. The diffusing computation is initiated at the neighbors of $\overline{v}$ because only these nodes are aware if $\overline{v}$ is used an intermediary node. The diffusing computations spread from $\overline{v}$'s neighbors to the network edge, invalidating false state at each node along the way. Then ACKs travel back from the network edge to the neighbors of $\overline{v}$, indicating that the diffusing computation is complete. Next, `purge` uses distance vector to recompute least cost paths invalidated by the diffusing computations.

1.3.4 The cpr Algorithm

`cpr`[2] is our third and final recovery algorithm. Unlike `2nd best` and `purge`, `cpr` requires that clocks across different nodes be loosely synchronized i.e., the maximum clock offset between any two nodes is bounded. Here we present `cpr` assuming all clocks are perfectly synchronized.

For each node, $i \in G$, `cpr` adds a time dimension to $\overrightarrow{min}_i$ and $dmatrix_i$, which `cpr` then uses to locally archive a complete history of values. Once the compromised node is discovered, the archive allows each node to rollback to a system snapshot from a time before $\overline{v}$ was compromised. `cpr` does so using diffusing computations. Then, `cpr` removes all $\overrightarrow{bad}$ and $\overrightarrow{old}$ state while updating stale distance values resulting from link cost changes that occurred between the time the snapshot was taken and the "current" time. This last step is executed by initiating a distance vector computation from the neigbhors of the compromised node.

1.4 Analysis of Algorithms

We derive communication complexity bounds for each algorithm over a synchronous communication model and fixed unit link weights. We find all three algorithms are bounded above by $O(mnd)$ – where d is the diameter, n is the number of nodes, and m the maximum out-degree of any node – in scenarios where link costs remain fixed. In scenarios where link costs can change, `cpr` incurs additional over-

[2]The name is an abbreviation for **C**heck**P**oint and **R**ollback.

head (not experienced by 2nd `best` and `purge`) because `cpr` must update stale state after rolling back. This overhead is not incurred by 2nd `best` and `purge` because neither algorithm rolls back in time. As a result, 2nd `best` and `purge` are still bounded $O(mnd)$ when link costs can change, while `cpr` is not. `cpr`'s upper bound, when link costs are not fixed, includes an additional term in its $O(mnd)$ bound.

SELF-NOTE: Summary of How I want to present the results:

1. `purge` is almost as good as `cpr` for fixed link costs even though: (a) the conditions are ideal for `cpr` and (b) `cpr` assumes perfectly syncrhonized clocks.
2. `purge` is close or better that `cpr` when there are link cost changes, plus `cpr` requires synchronized clocks and the frequency of when snapshots are taken needs to be set.
3. 2nd `best` suffers from routing loops and therefore the count-to-∞ problem.

1.5 Evaluation

In this section, we use simulations to characterize the performance of each of our three recovery algorithms in terms of message and time overhead. Our goal is to illustrate the relative performance of our recovery algorithms over different topology types (e.g., Erdös-Rényi graphs, Internet-like graphs) and different network conditions (e.g., fixed link costs, changing link costs). We evaluate recovery after a single compromised node has distributed false routing state.

We build a custom simulator with a synchronous communication model: nodes send and receive messages at fixed epochs. In each epoch, a node receives a message from all its neighbors and performs its local computation. In the next epoch, the node sends a message (if needed). All algorithms are deterministic under this communication model. The synchronous communication model, although simple, yields interesting insights into the performance of each of the recovery algorithms. Evaluation of our algorithms using a more general asynchronous communication model is currently under investigation. However, we believe an asynchronous implementation will demonstrate similar trends.

We simulate the following scenario:

1. Before t', $\forall v \in V$ $\overrightarrow{min_v}$ and $dmatrix_v$ are correctly computed.

2. At time t', $\overline{v}$ is compromised and advertises a $\overrightarrow{bad}$ (a vector with a cost of 1 to *every* node in the network) to its neighboring nodes.

3. $\overrightarrow{bad}$ spreads for a specified number of hops (this varies by experiment). Variable k refers to the number of hops that $\overrightarrow{bad}$ has spread.

4. At time t, some node $v \in V$ notifies all $v \in adj(\overline{v})$ that $\overline{v}$ was compromised. [3]

The message and time overhead are measured in step (4) above. The pre-computation common to all three recovery algorithms, described in Section 1.3.1, is not counted towards message and time overhead.

1.5.1 Fixed Link Weight Experiments

In the next three experiments, we evaluate our recovery algorithms over different topology types in the case of fixed link costs.

We start with a simplified experiment setting to measure the message and time overhead incurred by each of the recovery algorithms. In particular, we consider Erdös-Rényi graphs with parameters n and p. Further, the link weight of each edge in the graph is set to 50. n is the number of graph nodes and p is the probability

[3]For `cpr` this node also indicates the time, t', $\overline{v}$ was compromised.

that link (i, j) exists where $i, j \in V$. We iterate over different values of k. For each k, we generate an Erdös-Rényi graph, $G = (V, E)$, with parameters n and p. Then we select a $v \in V$ uniformly at random and simulate the scenario described above, using v as the compromised node. In total we sample 20 unique nodes for each G. We set $n = 100$, $p = \{0.05, 0.15, 0.25, 0.25\}$, and let $k = \{1, 2, ...10\}$. Each data point is an average over 600 runs (20 runs over 30 topologies). We then plot the 90% confidence interval.

`purge` and `2nd best` message overhead increases with larger k. Larger k implies more paths to repair, and therefore increased messaging. For values of k greater than a graph's diameter, the message overhead remains constant, as expected.

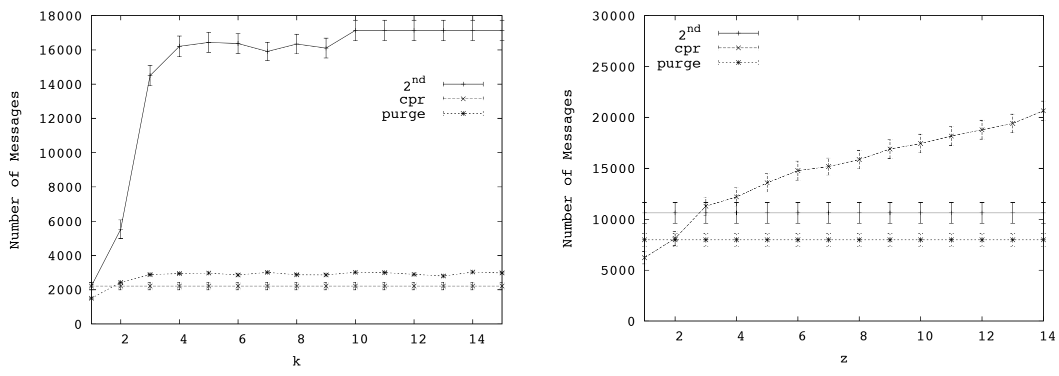
1.5.1.1 Experiment 2 - Erdös-Rényi Graphs with Fixed but Randomly Chosen Link Weights

The experimental setup is identical to Experiment 1 with one exception: link weights are selected uniformly at random between $[1, n]$ (rather than using fixed link weight of 50).

Figure **??** show the message overhead for different k for $p = .05$. We omit the figures for the other p values because they follow the same trend as $p = .05$ [13]. In striking contrast to Experiment 1, `purge` outperforms `2nd best` for all values of k. [4] `2nd best` performs poorly because the count-to-∞ problem: Table **??** shows the large average number of pairwise routing loops in this experiment, an indicator of the occurrence of count-to-∞ problem. No routing loops are found with `purge`. `cpr` performs well for the same reasons described in Section **??**.

In addition, we counted the number of epochs in which at least one pairwise routing loop existed. For `2nd best` (across all topologies), on average, all but the last

[4] In some cases (e.g., $k = 1$ for $p = .15$ and $p = .50$) `2nd best` performs better than `purge`. In these cases, `2nd best` has few routing loops [13].



(a) sim 1. (b) sim 2.

Figure 1.1. Rollback simulations

three timesteps had at least one routing loop. This suggests that the count-to-∞ problem dominates the cost for 2nd `best`.

1.5.2 Link Weight Change Experiments

So far, we have evaluated our algorithms over different topologies with fixed link costs. We found that `cpr` outperforms the other algorithms because `cpr` removes false routing state with a single diffusing computation, rather than using an iterative process like 2nd `best` and `purge`. In the next two experiments we evaluate our algorithms over graphs with changing link costs. We introduce link cost changes between the time $\overline{v}$ is compromised and when $\overline{v}$ is discovered (e.g. during $[t', t]$). In particular, there are λ link cost changes per timestep, where λ is deterministic. To create a link cost change event, we modify links uniformly at random (except for all $(v, \overline{v})$ links). The new link cost is selected uniformly at random from $[1, n]$.

1.5.2.1 Experiment 5

In this experiment we study the trade-off between message overhead and storage overhead for `cpr`. To this end, we vary the frequency at which `cpr` checkpoints and fix the interval $[t', t]$. Otherwise, our experimental setup is the same as Experiment 4.

Due to space constraints, we only display a single plot. Figure ?? shows the results for an Erdös-Rényi graph with link weights selected uniformly at random between $[1, n]$, $n = 100$, $p = .05$, $\lambda = 4$ and $k = 2$. We plot message overhead against the number of timesteps `cpr` must rollback, z. `cpr`'s message overhead increases with larger z because as z increases there are more link cost change events to process. 2^{nd} `best` and `purge` have constant message overhead because they operate independent of z.

We conclude that as the frequency of `cpr` snapshots decreases, `cpr` incurs higher message overhead. Therefore, when choosing the frequency of checkpoints, the trade-off between storage and message overhead must be carefully considered.

1.5.3 Summary

Our results show that for graphs with fixed link costs, `cpr` yields the lowest message and time overhead. `cpr` benefits from removing false state with a single diffusing computation. However, `cpr` has storage overhead, requires loosely synchronized clocks, and requires the time $\overline{v}$ was compromised be identified.

2^{nd} `best`'s performance is determined by the count-to-∞ problem. In this case of Erdös-Rényi graphs with fixed unit link weights, the count-to-∞ problem was minimal, helping 2^{nd} `best` perform better than `purge`. `purge` avoids the count-to-∞ problem by first globally invalidating false state. Therefore in cases where the count-to-∞ problem is significant, `purge` outperforms 2^{nd} `best`.

When considering graphs with changing link costs, `cpr`'s performance suffers because it must process all valid link cost changes that occurred since $\overline{v}$ was compromised. Meanwhile, 2^{nd} `best` and `purge` make use of computations that followed the injection of false state, that do not depend on false routing state. However, 2^{nd} `best`'s performance degrades because of the count-to-∞ problem. `purge` eliminates the count-to-∞ problem and therefore yields the best performance over topologies with changing link costs.

Finally, we found that an additional challenge with `cpr` is setting the parameter which determines the checkpoint frequency. More frequent checkpointing yields lower message and time overhead at the cost of more storage overhead. Ultimately, application-specific factors must be considered when setting this parameter.

1.6 Related Work

To the best our knowledge no existing approach exist to address recovery from false routing state in distance vector routing. However, our problem is similar to that of recovering from malicious but committed database transactions. Liu et al. [2] and Ammann et al [17] develop algorithms to restore a database to a valid state after a malicious transaction has been identified. `purge`'s algorithm to globally invalidate false state can be interpreted as a distributed implementation of the dependency graph approach by Liu et al. [17]. Additionally, if we treat link cost change events that occur after the compromised node has been discovered as database transactions, we face a similar design decision as in [2]: do we wait until recovery is complete before applying link cost changes or do we allow the link cost changes to execute concurrently?

Database crash recovery [19] and message passing systems [10] both use snapshots to restore the system in the event of a failure. In both problem domains, the snapshot algorithms are careful to ensure snapshots are globally consistent. In our setting, consistent global snapshots are not required for `cpr`, since distance vector routing only requires that all initial distance estimates be nonnegative.

Jefferson [15] proposes a solution to synchronize distributed systems called Time Warp. Time Warp is a form of optimistic concurrency control and, as such, occasionally requires rolling back to a checkpoint. Time Warp does so by “unsending” each message sent after the time the checkpoint was taken. With our `cpr` algorithm, a node does not need to explicitly "unsend" messages after rolling back. Instead, each node sends its $\overrightarrow{min}$ taken at the time of the snapshot, which implicitly undoes the effects of any messages sent after the snapshot timestamp.

1.7 Conclusions and Future Work

In this paper, we developed methods for recovery in scenarios where a malicious node injects false state into a distributed system. We studied an instance of this problem in distance vector routing. We presented and evaluated three new algorithms for recovery in such scenarios. Among our three algorithms, our results show that `cpr` – a checkpoint-rollback based algorithm – yields the lowest message and time overhead over topologies with fixed link costs. However, `cpr` has storage overhead and requires loosely synchronized clocks. In the case of topologies with changing link costs, `purge` performs best by avoiding the problems that plague `cpr` and `2`$^{\text{nd}}$ `best`. Unlike `cpr`, `purge` has no stale state to update because `purge` does not rollback in time. The count-to-∞ problem results in high message overhead for `2`$^{\text{nd}}$ `best`, while `purge` eliminates the count-to-∞ problem by globally purging false state before finding new least cost paths.

As future work, we are interested in finding the worst possible false state a compromised node can inject. Some options include the minimum distance to all nodes (e.g., our choice for false state used in this paper), state that maximizes the effect of the count-to-∞ problem, and false state that contaminates a bottleneck link. We also would like to evaluate the effects of multiple compromised nodes on our recovery algorithms.

CHAPTER 2

??? BACKGROUND: SMART GRID AND PMU SENSORS ???

CHAPTER 3

PMU SENSOR PLACEMENT FOR MEASUREMENT ERROR DETECTION IN THE SMART GRID

3.1 Introduction

Question: move paragraph to Smart Grid Overview Chapter ???? Significant investments have been made to deploy phasor measurement units (PMUs) on electric power grids worldwide. PMUs provide *synchronized* voltage and current measurements at a sampling rate orders of magnitude higher than the status quo: 10 to 60 samples per second rather than one sample every 1 to 4 seconds. This allows system operators to directly measure the state of the electric power grid in real-time, rather than rely on imprecise state estimation. Consequently, PMUs have the potential to enable an entirely new set of applications for the power grid: protection and control during abnormal conditions, real-time distributed control, postmortem analysis of system faults, advanced state estimators for system monitoring, and the reliable integration of renewable energy resources [5].

Question: move paragraph to Smart Grid Overview Chapter ???? An electric power system consists of a set of buses – an electric substation, power generation center, or aggregation of electrical loads – and transmission lines connecting those buses. The state of a power system is defined by the voltage phasor – the magnitude and phase angle of electrical sine waves – of all system buses and the current phasor of all transmission lines. PMUs placed on buses provide real-time measurements of these system variables. However, because PMUs are expensive, they cannot be deployed on all system buses [3][8]. Fortunately, the voltage phasor at a system bus can, at times, be determined (termed *observed* in this paper) even when a PMU is not placed at that bus, by applying Ohm's and Kirchhoff's laws on the measurements taken by a PMU placed at some nearby system bus [3][6]. Specifically, with correct placement of enough PMUs at a subset of system buses, the entire system state can be determined.

Question: only talk about MaxObserve and MaxObserve-XV ???? In this chapter, we study two sets of PMU placement problems. The first problem set consists of FullObserve and MaxObserve, and considers maximizing the observability of the network via PMU placement. FullObserve considers the minimum number of PMUs needed to observe all system buses, while MaxObserve considers the maximum number of buses that can be observed with a given number of PMUs. A bus is said to be *observed* if there is a PMU placed at it or if its voltage phasor can be estimated using Ohm's or Kirchhoff's Law. Although FullObserve is well studied [3, 6, 14, 18, 24], existing work considers only networks consisting solely of zero-injection buses, an unrealistic assumption in practice, while we generalize the problem formulation to include mixtures of zero and non-zero-injection buses. Additionally, our approach for analyzing FullObserve provides the foundation with which to present the other three new (but related) PMU placement problems.

The second set of placement problems considers PMU placements that support PMU error detection. PMU measurement errors have been recorded in actual systems [23]. One method of detecting these errors is to deploy PMUs "near" each other, thus enabling them to *cross-validate* each-other's measurements. FullObserve-XV aims to minimize the number of PMUs needed to observe all buses while insuring PMU cross-validation, and MaxObserve-XV computes the maximum number of observed buses for a given number of PMUs, while insuring PMU cross-validation.

We make the following contributions in this chapter:

- We formulate two PMU placement problems, which (broadly) aim at maximizing observed buses while minimizing the number of PMUs used. Our formulation extends previously studied systems by considering both zero and non-zero-injection buses.

- We formally define graph-theoretic rules for PMU cross-validation. Using these rules, we formulate two additional PMU placement problems that seek to maximize the observed buses while minimizing the number of PMUs used under the condition that the PMUs are cross-validated.

- We prove that all four PMU placement problems are NP-Complete. This represents our most important contribution.

- Given the proven complexity of these problems, we evaluate heuristic approaches for solving these problems. For each problem we describe a greedy algorithm, and prove that each greedy algorithm has polynomial running time.

- Using simulations, we evaluate the performance of our greedy approximation algorithms over synthetic and actual IEEE bus systems. We find that the greedy algorithms yield a PMU placement that is, on average, within 97% optimal. Additionally, we find that the cross-validation constraints have limited effects on observability: on average our greedy algorithm that places PMUs according to the cross-validation rules observes only 5.7% fewer nodes than the same algorithm that does not consider cross-validation.

The rest of this chapter is organized as follows. In Section 3.2 we introduce our modeling assumptions, notation, and observability and cross-validation rules. In Section 3.3 we formulate and prove the complexity of our four PMU placement problems. Section 3.4 presents the approximation algorithms for each problem and Section 3.5 considers our simulation-based evaluation. We conclude with a review of related work (Section **??**) and concluding remarks (Section 3.7).

3.2 Preliminaries

In this section we introduce notation and underlying assumptions (Section 3.2.1), and define our observability (Section 3.2.2) and cross-validation (Section 3.2.3) rules.

Question: Maybe merge Observability Rules and Cross-Validation Rules Sections ????

3.2.1 Assumptions, Notation, and Terminology

We model a power grid as an undirected graph $G = (V, E)$. Each $v \in V$ represents a bus. $V = V_Z \cup V_I$, where V_Z is the set of all zero-injection buses and V_I is the set of all non-zero-injection buses. A bus is zero-injection if it has no load nor generator [26]. All other buses are non-zero-injection, which we refer to as injection buses. Each $(u, v) \in E$ is a transmission line connecting buses u and v.

Consistent with the conventions in [3, 6, 7, 18, 24, 25], we assume: PMUs can only be placed on buses and a PMU on a bus measures the voltage phasor at the bus and the current phasor of all transmission lines connected to it. For convenience, we refer to any bus with a PMU as a *PMU node*.

For $v \in V$ define let $\Gamma(v)$ be the set of v's neighbors in G. A PMU placement $\Phi_G \subseteq V$ is a set of nodes at which PMUs are placed, and $\Phi_G^R \subseteq V$ is the set of observed nodes for graph G with placement Φ_G (see definition of observability below).

3.2.2 Observability Rules

We use the simplified observability rules stated by Brueni and Heath [6]:

1. **Observability Rule 1 (O1)**. *If node v is a PMU node, then $v \cup \Gamma(v)$ is observed.*

2. **Observability Rule 2 (O2)**. *If a zero-injection node, v, is observed and $\Gamma(v) \backslash \{u\}$ is observed for some $u \in \Gamma(v)$, then $v \cup \Gamma(v)$ is observed.*

Since O2 only applies with zero-injection nodes, the number of zero-injection nodes can greatly affect system observability.

3.2.3 Cross-Validation Rules

Cross-validation formalizes the intuitive notion of placing PMUs "near" each other to allow for measurement error detection. For convenience, we say a PMU is cross-validated even though it is actually the PMU data at a node that is cross-validated. A PMU is *cross-validated* if one of the rules below is satisfied [23]:

1. **Cross-Validation Rule 1 (XV1)**. *If two PMU nodes are adjacent, then the PMUs cross-validate each other.*

2. **Cross-Validation Rule 2 (XV2)**. *If two PMU nodes have a common neighbor, then the PMUs cross-validate each other.*

XV1 derives from the fact that both PMUs are measuring the current phasor of the transmission line connecting the two PMU nodes. XV2 is more subtle. Using the notation specified in XV2, when computing the voltage phasor of an element in $\Gamma(u) \cap \Gamma(v)$ the voltage equations include variables to account for measurement error (e.g., angle bias) [22]. When the PMUs are two hops from each other, there are more equations than unknowns, allowing for measurement error detection. Otherwise, the number of unknown variables exceeds the number of equations, which eliminates the possibility of detecting measurement errors [22].

3.3 Four NP-Complete PMU Placement Problems

In this section we first define four PMU placement problems and then provide a high-level description of the proof strategy we use to prove each problem is NP-Complete. In all four problems defined in this paper, we are only concerned with computing the voltage phasors of each bus (i.e., observing the buses). Using the values of the voltage phasors, Ohm's Law can be easily applied to compute the current phasors of each transmission line. Also, we consider networks with both injection and zero-injection buses.

3.3.1 Problem Statements

Here we briefly define each of our four PMU placement problems: FullObserve, MaxObserve , MaxObserve-XV, and FullObserve-XV.

FullObserve Decision Problem:

Instance: Graph $G = (V, E)$ where $V = V_Z \cup V_I$, $V_Z \neq \emptyset$, k PMUs such that $k \geq 1$.

Question: Is there a Φ_G such that $|\Phi_G| \leq k$ and $\Phi_G^R = V$?

MaxObserve Decision Problem:

Instance: Graph $G = (V, E)$ where $V = V_Z \cup V_I$, k PMUs such that $1 \leq k < k^*$.

Question: For a given $m < |V|$, is there a Φ_G such that $|\Phi_G| \leq k$ and $m \leq |\Phi_G^R| < |V|$?

FullObserve-XV Decision Problem:

Instance: Graph $G = (V, E)$ where $V = V_Z \cup V_I$, k PMUs such that $k \geq 1$.

Question: Is there a Φ_G such that $|\Phi_G| \leq k$ and $\Phi_G^R = V$ under the condition that each $v \in \Phi_G$ is cross-validated?

MaxObserve-XV Decision Problem:

Instance: Graph $G = (V, E)$ where $V = V_Z \cup V_I$, k PMUs such that $1 \leq k < k^*$, and some $m < |V|$.

Question: Is there a Φ_G such that $|\Phi_G| \leq k$ and $m \leq |\Phi_G^R| < |V|$ under the condition that each $v \in \Phi_G$ is cross-validated?

Theorem 1. MaxObserve, MaxObserve-XV, FullObserve *and* FullObserve-XV *are all NP-Complete.*

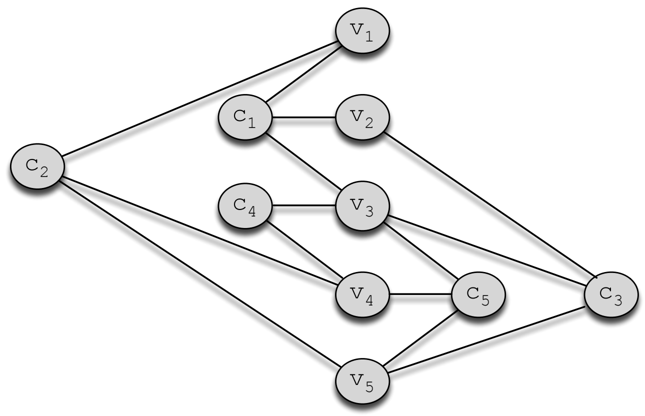
3.3.2 Overview of NPC Proof Strategy

In this section, we outline the proof strategy we used in each of NP-Completeness proofs. Due to space constraints we omit the actual NPC proofs. Our proofs follow a similar structure proposed by Brueni and Heath [6]. The authors prove NP-Completeness by reduction from planar 3-SAT (P3SAT). A 3-SAT formula, ϕ, is a boolean formula in conjunctive normal form (CNF) such that each clause contains at most 3 literals. For any 3-SAT formula ϕ with the sets of variables $\{v_1, v_2, \ldots, v_r\}$ and clauses $\{c_1, c_2, \ldots, c_s\}$, $G(\phi)$ is the bipartite graph $G(\phi) = (V(\phi), E(\phi))$ defined as follows:

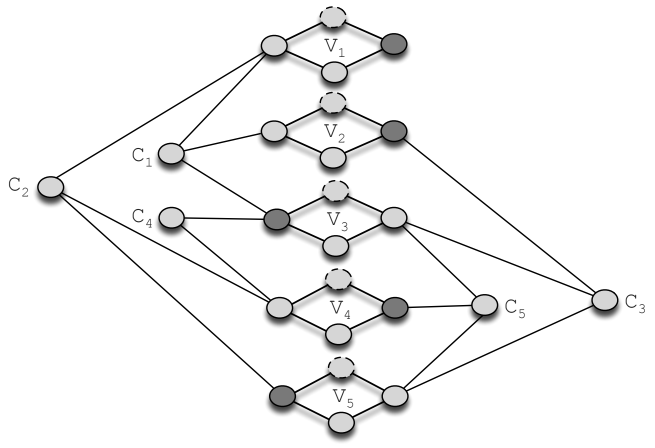
$$\begin{aligned} V(\phi) &= \{v_i \mid 1 \leq i \leq r\} \cup \{c_j \mid 1 \leq j \leq s\} \\ E(\phi) &= \{(v_i, c_j) \mid v_i \in c_j \;\; or \;\; \overline{v_i} \in c_j\}. \end{aligned}$$

Note that edges pass only between v_i and c_j nodes, and so the graph is bipartite. P3SAT is a 3-SAT formula such that $G(\phi)$ is planar [16]. For example, P3SAT formula $\varphi = (\overline{v_1} \vee v_2 \vee v_3) \wedge (\overline{v_1} \vee \overline{v_4} \vee v_5) \wedge (\overline{v_2} \vee \overline{v_3} \vee \overline{v_5}) \wedge (v_3 \vee \overline{v_4}) \wedge (\overline{v_3} \vee v_4 \vee \overline{v_5})$ has graph $G(\varphi)$ shown in Figure 3.1(a). Discovering a satisfying assignment for P3SAT is an NPC problem, and so it can be used in a reduction to prove the complexity of the problems we address here.

Following the approach in [6], for P3SAT formula, ϕ, we replace each variable node and each clause node in $G(\phi)$ with a specially constructed set of nodes, termed a *gadget*. In this work, all variable gadgets will have the same structure, and all clause gadgets have the same structure (that is different from the variable gadget structure), and we denote the resulting graph as $H(\phi)$. In $H(\phi)$, each *variable* gadget has a subset of nodes that semantically represent assigning "True" to that variable, and a subset of nodes that represent assigning it "False". When a PMU is placed at one of these nodes, this is interpreted as assigning a truth value to the P3SAT variable corresponding with that gadget. Thus, we use the PMU placement to determine a



(a) $G(\varphi) = (V(\varphi), E(\varphi))$ formed from φ in Equation (**??**).



(b) Graph formed using variable gadget from ...

Figure 3.1. Example

consistent truth value for each P3SAT variable. Also, clause gadgets are connected to variable gadgets at either "True" or "False" (but never both) nodes, in such a way that the clause is satisfied if and only if *at least one* of those nodes has a PMU.

While the structure of our proofs is adapted from [6], the variable and clause gadgets we use to correspond to the P3SAT formula are novel, thus leading to a different set of proofs. Our work here demonstrates how the work in [6] can be extended, using new variable and clause gadgets, to address a wide array of PMU placement problems.

3.4 Approximation Algorithms

Because all four placement problems are NPC, we propose greedy approximation algorithms for each problem, which iteratively add a PMU in each step to the node that observes the maximum number of new nodes. We present two such algorithms, one which directly addresses MaxObserve (`greedy`) and the other MaxObserve-XV (`xvgreedy`). `greedy` and `xvgreedy` can easily be used to solve FullObserve

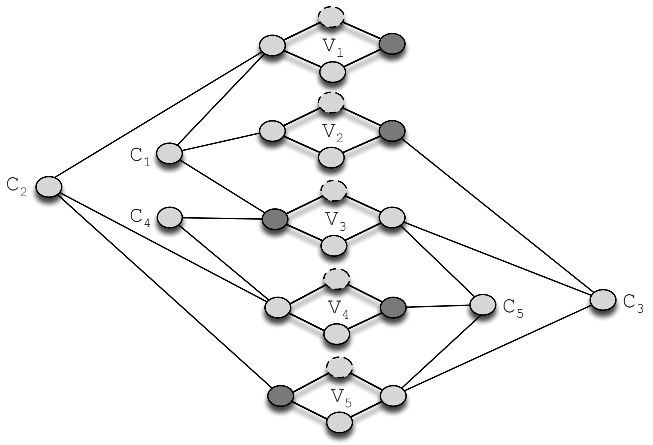
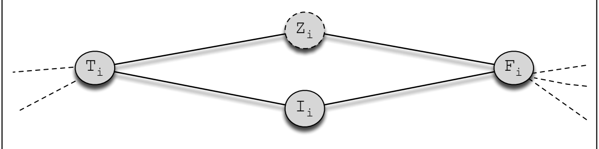
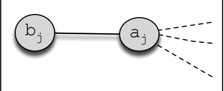


Figure 3.2. Graph $G = (V, E) = H_1(\varphi)$ formed from φ formula in Theorem **??** proof. Nodes with a dashed border are zero-injection nodes.



(a) Variable gadget V_i used in Theorem **??** and Theorem **??**.



(b) Clause gadget C_j used in Theorem **??**.

Figure 3.3. Gadgets used in Theorem **??** and Theorem **??**. Z_i in Figure (a) is the only zero-injection node. The dashed edges in Figure (a) are connections to clause gadgets. Likewise, the dashed edges in Figure (b) are connections to variable gadgets.

and FullObserve-XV, respectively, by selecting the appropriate k value to ensure full observability.

`greedy` **Algorithm**. We start with $\Phi = \emptyset$. At each iteration, we add a PMU to the node that results in the observation of the maximum number of new nodes. The algorithm terminates when all PMUs are placed. [1]

`xvgreedy` **Algorithm**. `xvgreedy` is almost identical to `greedy`, except that PMUs are added in pairs such that the selected pair observe the maximum number of nodes under the condition that the PMU pair satisfy one of the cross-validation rules.

[1]The same greedy algorithm is proposed by Aazami and Stilp [1].

3.5 Simulation Study

Topologies. We evaluate our approximation algorithms with simulations over synthetic topologies generated using real portions of the North American electric power grid (i.e., IEEE bus systems 14, 30, 57, and 118) as templates. [2]. The bus system number indicates the number of nodes in the graph (e.g., bus system 57 has 57 nodes). It is standard practice in the literature to only use single IEEE bus systems [3, 7, 18, 24]. We follow this precedent but do not present these results because they are consistent with the trends found using synthetic topologies. Instead, we focus on synthetic topologies because, unlike simulations using single IEEE bus systems, we can establish the statistical significance of greedy approximations.

Since observability is determined by the connectivity of the graph, we use the *degree distribution* of IEEE topologies as the template for generating our synthetic graphs. A synthetic topology is generated from a given IEEE graph by randomly "swapping" edges in the IEEE graph. Specifically, we select a random $v \in V$ and then pick a random $u \in \Gamma(v)$. Let u have degree d_u. Next, we select a random $w \notin \Gamma(v)$ with degree $d_w = d_u - 1$. Finally, we remove edge (v, u) and add (v, w), thereby preserving the node degree distribution. We continue this swapping procedure until the original graph and generated graph share *no edges*, and then return the resulting graph.

Evaluation Methods. We are interested in evaluating how close our algorithms are to the optimal PMU placement. Thus, when computationally possible (for a given k) we use brute-force algorithms to iterate over all possible placements of k PMUs in a given graph and select the best PMU placement. When the brute-force algorithm is computationally infeasible, we present only the performance of the greedy algorithm.

[2]http://www.ee.washington.edu/research/pstca/

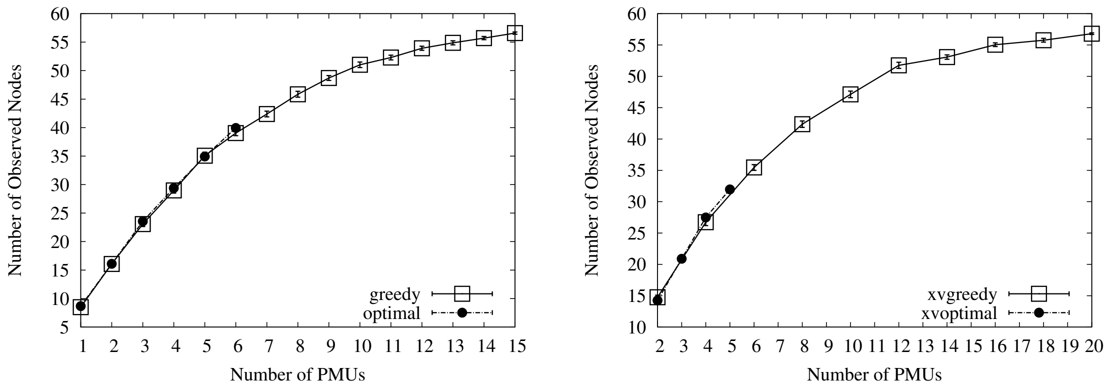
In what follows, the output of the brute-force algorithm is denoted `optimal`, and when we require cross-validation it is denoted `xvoptimal`.

Simulation. We vary the number of PMUs and determine the number of observed nodes in the synthetic graph. Each data point is generated as follows. For a given number of PMUs, k, we generate a graph, place k PMUs on the graph, and then determine the number of observed nodes. We continue this procedure until $[0.9(\overline{x}), 1.1(\overline{x})]$ – where $\overline{x}$ is the mean number of observed nodes using k PMUs – falls within the 90% confidence interval.

In addition to generating a topology, for each synthetic graph we determined the members of V_I, V_Z. These nodes are specified for the original graphs in the IEEE bus system database. Thus, we randomly map each node in the IEEE graph to a node in the synthetic graph with the same degree, and then match their membership to either V_I or V_Z.

Due to space constraints, we only show plots for solving MaxObserve and MaxObserve-XV using synthetic graphs based on IEEE bus 57. The number of nodes observed given k, using `greedy` and `optimal`, are shown in Figure 3.4(a), and Figure 3.4(b) shows this number for `xvgreedy` and `xvoptimal`. Both plots include the 90% confidence intervals. Note that results for synthetic graphs generated using IEEE bus 14, 30, and 118 yield the same trends.

Our greedy algorithms perform well. On average, `greedy` is within 98.6% of `optimal`, is never below 94% of `optimal`, and in most cases gives the optimal result. Likewise, `xvgreedy` is never less than 94% of `xvoptimal` and on average is within 97% of `xvoptimal`. In about about half the cases `xvgreedy` gives the optimal result. These results suggest that despite the complexity of the problems, a greedy approach can return high-quality results. Note, however, that these statistics do not include performance when k is large. It is an open question whether `greedy` and `xvgreedy` would do well for large k.



(a) `greedy` vs `optimal` (b) `xvgreedy` vs `xvoptimal`

Figure 3.4. Mean number of observed nodes over synthetic graphs based on IEEE bus 57 when varying number of PMUs. The 90% confidence interval is shown.

Surprisingly, when comparing our results with and without the cross-validation requirement, we find that the cross-validation constraints have little effect on the number of observed nodes for the same k. Our experiments show that on average `xvoptimal` observed only 5% fewer nodes than `optimal`. Similarly, on average `xvgreedy` observes 5.7% fewer nodes than `greedy`. This suggests that the cost of imposing the cross-validation requirement is low, with the clear gain of ensuring PMU correctness across the network.

3.6 Related Work

FullObserve is well-studied [3, 6, 14, 18, 24]. Haynes et al. [14] and Brueni and Heath [6] both prove FullObserve is NPC. However, their proofs make the unrealistic assumption that all nodes are zero-injection. We drop this assumption and thereby generalize their NPC results for FullObserve. Additionally, we leverage the proof technique from Brueni and Heath [6] in all four of our NPC proofs, although our proofs differ considerably in their details.

In the power systems literature, Xu and Abur [24, 25] use integer programming to solve FullObserve, while Baldwin et al. [3] and Mili et al. [18] use simulated annealing to solve the same problem. All of these works allow nodes to be either zero-injection or non-zero-injection. However, these papers make no mention that FullObserve is NPC, i.e., they do not characterize the fundamental complexity of the problem.

Aazami and Stilp [1] investigate approximation algorithms for FullObserve. They derive a hardness approximation threshold of $2^{\log^{1-\epsilon} n}$. Also they prove that in the worst case, `greedy` from Section 3.4 does no better $\Theta(n)$ of the optimal solution. However, this approximation ratio assumes that all nodes are zero-injection.

Chen and Abur [7] and Vanfretti et al. [23] both study the problem of bad PMU data. Chen and Abur [7] formulate their problem differently than FullObserve-XV and MaxObserve-XV. They consider fully observed graphs and add PMUs to the system to make all existing PMU measurements non-critical (a critical measurement is one in which the removal of a PMU makes the system no longer fully observable). Vanfretti et al. [23] define the cross-validation rules used in this paper. They also derive a lower bound on the number of PMUs needed to ensure all PMUs are cross-validated and the system is fully observable.

3.7 Conclusions and Future Work

In this chapter, we formulated four PMU placement problems and proved that each one is NPC. Consequently, future work should focus on developing approximation algorithms for these problems. As a first step, we presented two simple greedy algorithms: `xvgreedy` which considers cross-validation and `greedy` which does not. Both algorithms iteratively add PMUs to the node which observes the maximum of number of nodes.

Using simulations, we found that our greedy algorithms consistently reached close-to-optimal performance. We also found that cross-validation had a limited effect on observability: for a fixed number of PMUs, `xvgreedy` and `xvoptimal` observed only 5% fewer nodes than `greedy` and `optimal`, respectively. As a result, we believe imposing the cross-validation requirement on PMU placements is advised, as the benefits they provide come at a low marginal cost.

There are several topics for future work. The success of the greedy algorithms suggests that bus systems have special topological characteristics, and we plan to investigate their properties. Additionally, we intend to implement the integer programming approach proposed by Xu and Abur [24] to solve FullObserve. This would provide valuable data points to measure the relative performance of `greedy`.

CHAPTER 4

RECOVERY FROM LINK FAILURES IN A SMART GRID COMMUNICATION NETWORK

APPENDIX

THE FIRST APPENDIX TITLE

…

BIBLIOGRAPHY

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

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

[3] 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.

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

[5] 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.

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

[7] 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.

[8] 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.

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

[10] 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.

[11] 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).

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

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

[14] 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.

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

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

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

[18] 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.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

[26] 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.