INTRODUCTION

Communication network components (routers, links, and sensors) fail. These failures can cause widespread network service disruption and outages, and potentially critical errors for network applications. In this thesis, we examine *how networks – traditional networks and networked cyber-physical systems, such as the smart grid – can be made more robust to component failure.*

We propose on-demand recovery algorithms for distributed network algorithms that optimize for control message overhead and convergence time, and preplanned approaches to recovery for electric power grid applications, where reliability is key. 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. We refer to modern and future electric power grids that automate power grid operations using sensors and wide-area communication as the *smart grid.*

0.1 Thesis Overview

0.1.1 Component Failure in Communication Networks

In this thesis, we consider three separate but related problems: node (i.e., switch or router) failure in traditional networks such as the Internet or wireless sensor networks, the failure of critical sensors that measure voltage and current throughout the smart grid, and link failures in a smart grid communication network. For distributed network algorithms, a malicious or misconfigured node can inject and spread incorrect state throughout the distributed system. Such false state can degrade the performance of the network or render it unusable. For example, in 1997 a significant portion of Internet traffic was routed through a single misconfigured router that had

spread false routing state to several Internet routers. As a result, a large potion of the Internet became inoperable for several hours [36].

In a smart grid, especially, component failure can be catastrophic. For example, if smart grid sensors or links in its supporting communication network fail, smart grid applications can make incorrect decisions and take corresponding (incorrect) actions. Critical smart grid applications required to operate and manage a power grid are especially vulnerable to such failures because typically these applications have strict data delivery requirements, needing both ultra low latency and assurance that data is received correctly. In the worst case, component failure can lead to a cascade of power grid failures like the August 2003 blackout in the USA [?] and the recent power grid failures in India [46].

0.1.2 Approaches to Making Networks More Robust to Failures

For most distributed systems, recovery algorithms operate on-demand (as opposed to being preplanned) because algorithm and system state is typically distributed throughout the network of nodes. As a result, fast convergence time and low control message overhead are key requirements for efficient recovery from component failure. In order to make the problem of on-demand recovery in a distributed system 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).

In the first technical chapter of this thesis, we design, develop, and evaluate three different approaches for correctly recovering from the injection of false distance vector 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, causing other nodes to (incorrectly) route

via the misconfigured node, 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.

The second and third thesis chapters consider robustness from component failure specifically in the context of the smart grid. Because reliability is a key requirement for the smart grid, we focus on preplanned approaches to failure recovery.

In our second thesis chapter, 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. We formulate a set of problems that consider PMU measurement errors, which have been observed in practice. Specifically, we specify four 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.

In our final technical thesis chapter, we present the initial design for algorithms that provide recovery from link failures in a smart grid communication network. ~~Broadly speaking, these recovery algorithms aim to maximize the long-term survivability of the smart~~ grid. The recovery problem divides into two parts: (a) link failure detection and (b) algorithms for pre-computing backup multicast trees. To address (a), we sketch the design of a link-failure detection and reporting mechanisms that use OpenFlow to detect link failures when and where they occur *inside* the network. OpenFlow is an open source framework that cleanly separates the control and

data planes for use in centralized network management and control. For part (b), we propose initial outlines for a set of algorithms that precompute backup multicast trees to be installed after a link failure. As future work, we plan to implement these algorithms in Openflow and evaluate them.

0.2 Thesis Contributions

The main contributions of this thesis are:

- We design, develop, and evaluate three different algorithms – 2ND-BEST, PURGE, and CPR – for correctly recovering from the injection of false routing state in distance vector routing. 2ND-BEST performs localized state invalidation, followed by network-wide recovery using the traditional distance vector algorithm. PURGE first globally invalidates false state and then uses distance vector routing is used to recompute distance vectors. CPR takes and stores local routing table snapshots at each router, and then uses a rollback mechanism to implement recovery. We prove the correctness of each algorithm for scenarios of single and multiple compromised nodes.

- We use simulations and analysis to evaluate 2ND-BEST, PURGE, and CPR in terms of control message overhead and convergence time. We find that 2ND-BEST performs poorly due to routing loops. Over topologies with fixed link costs, PURGE performs nearly as well as CPR even though our simulations and analysis assume near perfect conditions for CPR. Over more realistic scenarios in which link weights can change, we find that PURGE yields lower message complexity and faster convergence time than CPR and 2ND-BEST.

- We define four PMU placement problems, three of which are completely new, that place PMUs at a subset of electric power grid buses. Two PMU placement problems consider measurement error detection by requiring PMUs to be placed

"near" each other to allow for their measurements to be cross-validated. For each PMU placement problem, we prove it is NP-Complete and propose a simple greedy approximation algorithm.

- We prove our greedy approximations for PMU placement are correct and give complexity bounds for each. Through simulations over synthetic topologies generated using real portions of the North American electric power grid as templates, we find that our greedy approximations yield results that are close to optimal: on average, within 97% of optimal. We also find that imposing our requirement of cross-validation to ensure PMU measurement error detection comes at small marginal cost: on average, only 5% fewer power grid buses are observed (covered) when PMU placements require cross-validation versus placements that do not.

- We propose initial sketches of algorithms for preplanned recovery from link failures in a smart grid communication network. Our proposed research divides into two parts: link failure detection and algorithms for pre-computing backup multicast trees. For the first part, we design algorithms that use OpenFlow to detect and report link failures when and where they occur, *inside* the network. To address the second part, we propose a set of algorithms, each of which computes backup multicast trees that are installed after a link failure. 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, minimizing the number of affected flows across all multicast trees, and minimizing the number of affected sink nodes across all multicast trees. These optimization criteria differ from those proposed in the literature.

0.3 Thesis Outline

The rest of this thesis proposal is organized as follows. We present algorithms for recovery from false routing state in distributed routing algorithms in Chapter 1. In Chapter 2 we formulate PMU placement problems that provide measurement error detection. Chapter 3 presents the algorithm sketches for efficient recovery from link failures in a smart grid communication network. We conclude by outlining our planned future work in Chapter 4.

our initial and proposed research

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 chapter, 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 [36]. 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 chapter, 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: 2ND-BEST, PURGE, and CPR. 2ND-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 2ND-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, derive communication complexity bounds, and evaluate its efficiency in terms of message overhead and convergence time via simulation. Our analysis and simulations show that when considering topologies in which link costs remain fixed, CPR outperforms both PURGE and 2ND-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 2ND-BEST. CPR performs poorly because, following rollback, it must process

the valid link cost changes that occurred since the false routing state was injected; 2ND-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 [5, 31] in distributed databases. Our problem is also similar to that of rollback in optimistic parallel simulation [28]. 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 chapter 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 chapter.

This chapter has six sections. In Section 1.2 we define the false-state-removal 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 message 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. The research described here has been published in [7] cite your ICDCS paper

1.2 Problem Formulation

We consider distance vector routing [9] 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 compromised node is provided by a different algorithm, and thus do not consider this problem in this paper. Examples of such algorithms include [20, 21, 23] in the context of wired networks and [40] 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.

1.3 Recovery Algorithms

In this section we propose three new recovery algorithms: 2ND-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 $V' = V - \{\overline{v}\}$, $E' = E - \{(\overline{v}, v_i) \mid v_i \in adj(\overline{v})\}$, and link weight function $w : E \rightarrow \mathbb{N}$. $\overrightarrow{min}_v$ and $dmatrix_v$ are computed via the algorithms discussed below $\forall v \in V'$.

[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.

Before we describe each recovery algorithm, we outline a preprocessing procedure common to all three recovery algorithms.

1.3.1 Preprocessing Algorithm

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 [19] 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 2^{nd} 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 neighbors of the compromised node.

---

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