1.4 Analysis of Algorithms

Here we summarize the results from our analysis. The detailed proofs can be found in our corresponding technical report [25]. Using a synchronous communication model, we derive communication complexity bounds for each algorithm. Our analysis assumes: a graph with unit link weights of 1, only a single node is compromised, and the compromised node falsely claims a cost of 1 to every node in the graph. For graphs with fixed link costs, we find that the communication complexity of all three algorithms is 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 the second part of our analysis, we consider graphs where link costs can change. Again, we assume a graph with unit link weights of 1 and a single compromised node that declares a cost of 1 to every node. Additionally, we let link costs increase between the time the malicious node is compromised and now. We assume that across all network links, the total increase in link weights is w units. We find that CPR incurs additional overhead (not experienced by 2ND-BEST and PURGE) because CPR must update stale state after rolling back. 2ND-BEST and PURGE avoid the issue of stale state because neither algorithm rolls back in time. As a result, the message complexity for 2ND-BEST and PURGE is still bounded by $O(mnd)$ when link costs can change, while CPR is not. CPR's upper bound becomes $O(mnd) + O(wn^2)$.

1.5 Simulation Study

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 topologies (e.g., Erdös-Rényi graphs, Internet-like graphs) and across different network conditions (e.g., topologies with fixed link costs, topologies with changing link costs, a single compromised node, and multiple compromised nodes).

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, described in Section 1.3.1, is not counted towards message and time overhead because all three recovery algorithms use this same procedure.

In order to keep this thesis proposal document brief, we only discuss a subset of our simulation results. We focus on a simplified scenario in which a single compromised node has distributed false routing state. We consider Erdös-Rényi graphs in cases where link costs remain fixed (Section 1.5.1) and link costs change (Section 1.5.2).

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

$k=1$	$k=2$	$k=3$	$k=4-10$
554	1303	9239	12641

Table 1.1. Average number pairwise routing loops for 2ND-BEST in simulation described in Section 1.5.1.

The corresponding technical report [25], discusses results using a richer simulation model that considers more realistic network conditions: an asynchronous communication model, Internet-like graphs generated using GT-ITM [1] and Rocketfuel [2], and multiple compromised nodes. We find that the trends discussed in this report hold when using these new topologies and additional simulation scenarios.

1.5.1 Simulations using Graphs with Fixed Link Weight

Here we evaluate our recovery algorithms, in terms of message and time overhead, using Erdös-Rényi graphs with fixed link weights. In particular, we consider Erdös-Rényi graphs with parameters n and p, where n is the number of graph nodes and p is the probability that link (i, j) exists where $i, j \in V$. Link weights are selected uniformly at random between $[1, n]$.

In order to establish statistical significance, we generate several Erdös-Rényi graphs to be used in our simulations. 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).

Figure 1.1 shows the message overhead as a function of k when $p = .05$. The 90% confidence interval is included in the plot. 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. 2ND-BEST performs

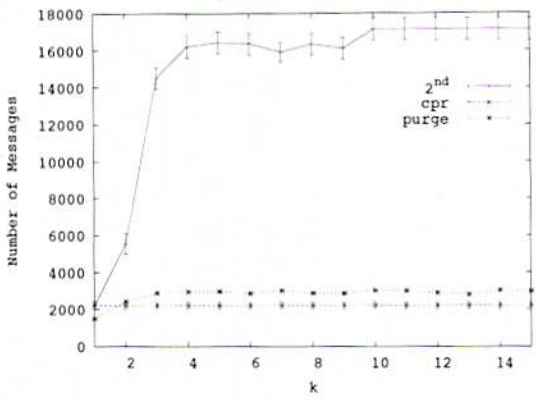


Figure 1.1. Message overhead as function of the number of hops false routing state has spread from the compromised node (k), over Erdös-Rényi graphs with fixed link weights. The Erdös-Rényi graphs are generated using $n = 100$, and $p = .05$, yielding an average diameter of 6.14.

poorly because of the count-to-∞ problem: Table 1.1 shows the large average number of pairwise routing loops, an indicator of the occurrence of count-to-∞ problem, 2ND-BEST encounters this simulation. 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 three timesteps had at least one routing loop. This suggests that the count-to-∞ problem dominates the cost for 2ND-BEST. In contrast, no routing loops are found with PURGE or CPR, as expected.

However, CPR's encouraging results should be interpreted with caution. CPR requires both loosely synchronized clocks and the time $\overline{v}$ was compromised to be identified, assumptions not required by 2ND-BEST nor PURGE. Furthermore, this first simulation scenario is ideal for CPR because fixed link costs ensures minimal stale state (i.e., residual $\overrightarrow{old}$ state) after CPR rolls back. The next two simulations present more challenging and less favorable conditions for CPR.

1.5.2 Simulations using Graphs with Changing Link Weights

In the next two simulations 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 $(v, \bar{v})$ links), where the new link cost is selected uniformly at random from $[1, n]$.

Effects of Link Cost Changes. Except for λ, our simulation setup is identical to the one in the previous section. We let $\lambda = \{1, 4, 8\}$. In order to isolate the effects of link costs changes, we assume that CPR checkpoints at each timestep.

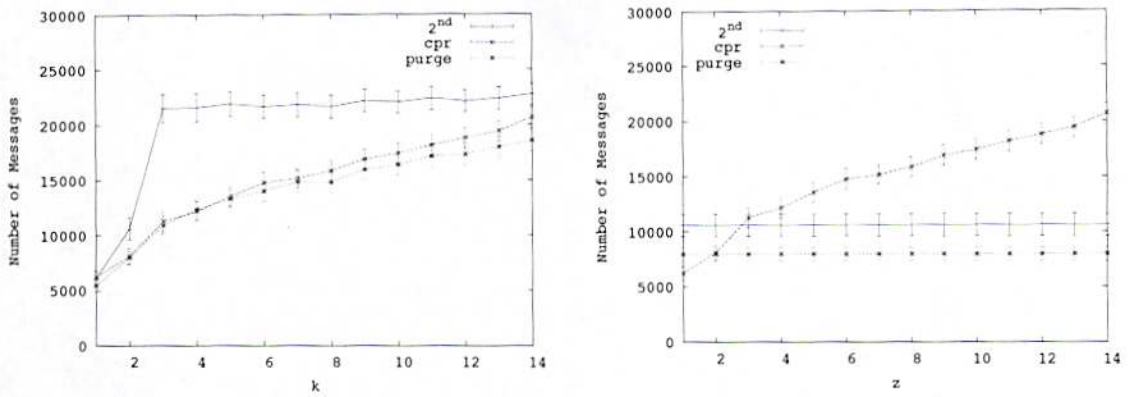
For the sake of brevity, we only show results for $n = 100, p = .05, \lambda = 4$ in Figure 1.2(a). [4] PURGE yields the lowest message overhead, but only slightly lower than CPR. CPR's message overhead increases with larger k because there are more link cost change events to process. After CPR rolls back, it must process all link cost changes that occurred in $[t', t]$. In contrast, 2ND-BEST and PURGE process some of the link cost change events during the interval $[t', t]$ as part of normal distance vector execution.

Our analysis further indicates that 2ND-BEST performance suffers because of the count-to-∞ problem. The gap between 2ND-BEST and the other algorithms shrinks as λ increases because link cost changes have a larger effect on message overhead as λ grows.

Effects of Varying Checkpoint Frequency. In this simulation, 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 simulation setup is the same as the one just described (under the title "Effects of Link Cost Changes").

For conciseness, we only display a single plot. Figure 1.2(b) shows the results for an Erdös-Rényi graph with link weights selected uniformly at random between $[1, n]$,

[4] Our simulations for different p values, yield the same trends. Refer to our technical report for more details [?].



(a) Message overhead as a function of the number of hops false routing state has spread (k) where $\lambda = 4$ link cost changes occur at each timestep.

(b) Message overhead as function of varying the checkpoint frequency, z. $z = 0$ implies checkpointing occurs at every timestep, $z = 1$ creates checkpoints every other timestamp, etc.

Figure 1.2. Section 1.5.2 plots. Both plots consider Erdös-Rényi graphs with changing link costs generated using $n = 100$ and $p = .05$. The average diameter of the generated graphs is 6.14.

$n = 100$, $p = .05$, $\lambda = 4$ 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. 2ND-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 of Simulation Results

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.

2ND-BEST's performance is determined by the count-to-∞ problem. PURGE avoids the count-to-∞ problem by first globally invalidating false state. Therefore in cases where the count-to-∞ problem is significant, PURGE outperforms 2ND-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, 2ND-BEST and PURGE make use of computations that followed the injection of false state, that do not depend on false routing state. However, 2ND-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. [5] and Ammann et al [31] 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. [31]. 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 [5]: 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 [35] and message passing systems [20] 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 non-negative.

Jefferson [28] 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 chapter, 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 2ND-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 2ND-BEST, while PURGE eliminates the count-to-∞ problem by globally purging false state before finding new least cost paths.

one challenging problem is to find

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.

CHAPTER 2

PMU SENSOR PLACEMENT FOR MEASUREMENT ERROR DETECTION IN THE SMART GRID

2.1 Introduction

This chapter considers placing electric power grid sensors, called phasor measurement units (PMUs), to enable measurement error detection. Significant investments have been made to deploy 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 [11].

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 [7][18]. 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 [7][12]. 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~~ calculated using Ohm's or Kirchhoff's Law. Although FULLOBSERVE is well studied [7, 12, 26, 34, 44], 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 [42]. 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 formula-