

# Continuous Containment and Local Stabilization in Path-vector Routing

Hongwei Zhang

Anish Arora

Technical report: OSU-CISRC-5/04-TR27

Department of Computer Science and Engineering

The Ohio State University, USA

{zhangho, anish}@cse.ohio-state.edu

**Abstract**— We formulate concepts that characterize network properties in the presence of continuously occurring faults, and we present CPV, a path-vector routing protocol that locally contains continuously occurring faults and locally stabilizes. Local containment enables CPV to protect distant nodes from being affected by faults. Local stabilization enables CPV to stabilize the network within time depending on the perturbation size after faults stop occurring. In CPV, the distance to which the state of a node propagates is proportional to the time the state remains valid. These properties are achieved by reacting to new fault only after first containing the response to the previous fault. In addition to analytically proving these properties, we evaluate CPV by simulating Internet-type networks with up to 75 autonomous systems; we observe that CPV reduces the number of fault-affected nodes by a factor of 71 and the network convergence time by a factor of 9.2 when compared with BGP.

**Keywords**— containment of continuously occurring faults, local stabilization, path-vector routing, BGP, Internet

## 1 Introduction

A well-known ideal in networking and distributed computing is the ability to withstand failure or compromise of one or more regions in a network without impacting a large part of the network [6, 19, 2]. To this end, formal models and mechanisms of fault containment have been proposed and/or used in practice [7, 3, 2, 17]. These models and mechanisms have largely focused on cases where faults stop occurring after certain moment in time, faults occur with low frequency, or faults assume specific patterns by which fault-occurrences can be predicted.

Nevertheless, faults may keep occurring with high frequencies, and the interval between faults may be shorter than the time taken for the network to stabilize. Moreover, complex interactions between different network components may generate unanticipated faults, especially when networks work in stressful conditions. Consequently, most existing mechanisms cannot guarantee fault containment in the presence of these high-frequency and unanticipated faults. One area where this issue remains to be addressed is inter-domain routing via path-vector protocol BGP in the Internet [19].

Under the Code Red/Nimda attack, for instance, memory overflow and BGP session reset cause routers to re-

peatedly fail-stop and rejoin at frequencies as high as once every minute [19]; even though instability-suppression timers (such as `MinRouteAdvertisementInterval` and `MinASOriginationInterval` [16]) and route-flap-damping are used, faults at some edge routers propagate across the whole Internet [19]. Such unbounded fault propagation decreases not only the availability of networks but also their stability and scalability.

To provide dependable services, therefore, networks must be able to contain the impact of high-frequency unanticipated faults. Moreover, networks should converge quickly once faults stop occurring, within time as a function of the degree of fault perturbation instead of the network size (we refer to this property as *local stabilization*). To this end, formal models that characterize network behaviors and mechanisms that guarantee fault containment in the presence of high-frequency unanticipated faults are desired.

**Related work.** The concepts of fault containment and local stabilization have been discussed in [7], [2], and [3]. But the definitions there are proposed only for the cases where faults stop or only state corruption can occur, thus they do not apply to cases where faults keep occurring at high frequencies and complex unanticipated faults may occur.

To locally contain faults in distance-vector routing, protocol LSRP [2] has been proposed. Yet LSRP guarantees fault containment and local stabilization only for scenarios where faults happen at low frequencies and networks get enough time to stabilize from one fault before another one occurs. Therefore, LSRP does not guarantee fault containment when faults keep occurring at high frequencies.

To improve the stability of BGP, instability-suppression timers and routing-flap-damping are used [16, 17]. Nevertheless, they only deal with faults of certain patterns (e.g., minimum inter-fault interval and fault predictability). Therefore, they do not guarantee fault containment in the presence of high-frequency unanticipated faults, as experienced in Internet [19] and observed in [2] and our simulation study in Section 6.

To improve BGP convergence speed after fault occurrence, various mechanisms have been proposed [15, 10, 4, 20, 13]. Nevertheless, these mechanisms do not focus on fault containment and thus cannot contain continuously occurring faults. On the other hand, these mechanisms

This work was partially sponsored by DARPA contract OSU-RF #F33615-01-C-1901, NSF grant NSF-CCR-9972368, an Ameritech Faculty Fellowship, and two grants from Microsoft Research.

can be used with those that guarantee fault containment to improve network stability and availability (to be discussed further in Section 7).

In [7], algorithms are proposed to contain a single state corruption during the stabilization of a spanning tree, but these algorithms do not deal with continuously occurring faults, node fail-stop, and node join. In [3], a broadcast protocol is proposed to contain observable variables in the presence of state corruptions, but the protocol allows for global propagation of internal protocol variables and is not “stability-adaptive”.

**Contributions of the paper.** To build the foundation for studying and to precisely characterize system properties in the presence of continuously occurring faults, we formulate the notions of perturbed node, contaminated node, perturbation size, contamination range,  $\mathcal{F}$ -containment, and  $\mathcal{F}$ -stabilization. These concepts are generally applicable to networking and distributed computing problems.

For the problem of path-vector routing, we design CPV, a path-vector routing protocol that contains continuously occurring faults and is locally stabilizing. In the presence of continuously occurring faults, the properties of CPV are as follows:

- The only nodes that are affected by a fault are those within  $O(p)$  distance from the fault-perturbed region, where  $p$  is the size of the perturbed region.
- The distance to which the state of a node propagates is proportional to the sojourn time of the state (i.e., the time for which the state remains valid), and as a result, the more unstable a node is, the shorter is the distance to which its state propagates.

Once faults stop occurring (either indefinitely or for a long enough period), the network converges to a legitimate state within  $O(\Gamma(p'))$  time, where  $p'$  is the perturbation size of the faults and  $\Gamma$  is a function dependent on the routing policy used in the network<sup>1</sup>.

CPV achieves these properties via the following design pattern. When a new fault occurs, before CPV generates a stabilization wave to correct the network routes, it first contains the effect of the stabilization wave resulting from the previous fault. The containment wave propagates faster than the stabilization wave to this end. To deal with the case where the containment wave is itself propagated in error (for instance, if yet another fault happens before the stabilization wave corresponding to the new fault is generated), CPV generates an undo-containment wave that propagates even faster than the containment wave. This wave is self-correcting. We find that this pattern is generally applicable to other networking and distributed computing problems.

We analytically evaluate the properties of CPV using the above concepts. We also evaluate CPV by simulating Internet-type networks with up to 75 autonomous systems (ASes); the results corroborate our analysis by

<sup>1</sup> $\Gamma$  is linear when the shortest-path-first route ranking policy is used.

showing that, in the presence of continuously occurring faults, CPV reduces the number of fault-affected nodes by a factor of 71 and the network convergence time by a factor of 9.2 when compared with BGP.

**Organization of the paper.** In Section 2, we present the network model, the protocol notation, and the fault model. We also briefly recall BGP. To simplify presentation, we discuss the design of CPV in Section 3 before formulating, in Section 4, the concepts of fault containment and local stabilization in the presence of continuously occurring faults. We analyze the properties of CPV in Section 5, and present the simulation results in Section 6. We further discuss CPV in Section 7 and make concluding remarks in Section 8.

## 2 Preliminaries

In this section, we present the network model, the protocol notation, and the fault model. We also briefly recall BGP.

**Network model.** A network  $G$  is an undirected graph  $(V, E, P)$ , where  $V$  is the set of nodes (i.e., BGP speakers),  $E$  is the set of links, and  $P$  is the function that defines the routing policies of each node. (In this paper, we only consider routing policy functions by which BGP converges.)  $V$  is divided into several subsets, each of which is an autonomous system (simply denoted as AS hereafter). Each node has a unique node-id, and all the nodes in the same AS have the same AS-id. For a node  $i$ , the id of its AS is denoted by  $i.as$ . For any two nodes  $i$  and  $j$ ,  $(i, j)$  is in  $E$  if  $i$  and  $j$  can communicate with each other directly, or if  $i$  and  $j$  are in the same AS.

Message transmission between nodes is reliable, and message passing delay across a link is bounded from above and from below by  $U$  and  $L$  respectively. There is a clock at each node; the ratio of clock rates between any two neighboring nodes is bounded from above by  $\alpha$ , but no extra constraint on the absolute values of clocks is enforced.

For clarity of presentation, we only consider one destination  $d$ , an address prefix representing a set of nodes in an AS  $d.as$ . (Our protocol readily applies when there are multiple destinations.)

**Protocol notation.** We write protocols using a variant of the Abstract Protocol notation [8]. At each node, the protocol consists of a finite set of variables and actions. Each action consists of three parts: guard, guard hold-time, and statement. For convenience, we associate a unique name with each action. Thus, an action has the following form:

$$\langle name \rangle :: \langle guard \rangle \xrightarrow{h} \langle statement \rangle$$

The guard is either a boolean expression over the protocol variables of the node or a message reception operation;  $h$  is the guard hold-time ( $h \geq 0$ ); the statement up-

dates zero or more protocol variables of the node and/or sends out some message(s). If  $h = 0$ , we write the action in the following form:

$$\langle name \rangle :: \langle guard \rangle \longrightarrow \langle statement \rangle$$

For an action whose guard is a message reception operation, its guard hold-time must be 0.

For an action named  $a$ , its guard hold-time is denoted by  $h.a$ . An action  $a$  is enabled at time  $t$  if the guard of  $a$  evaluates to true at  $t$ . An action  $a$  is executed at time  $t$  only if  $a$  is continuously enabled from time  $(t - h.a)$  to  $t$ . To execute an action, its statement is executed atomically.

**Fault model.** A node or a link is *up* if it functions correctly, and it is *down* if it fail-stops. We consider the following network faults: an up node or link can fail-stop and become down; a down node or link can become up and join the network; and the routing policies of a node can change. The interval between any two faults can be any non-negative value.

**Border Gateway Protocol.** BGP is a path-vector routing protocol (where a node maintains the complete AS-level path to each destination) used to coordinate routing among ASes in the Internet [16]. In BGP, UPDATE messages are passed between nodes to convey routing information. BGP UPDATES are route records that include the following attributes (among others):

<i>nlri</i>	: network layer reachability information (i.e., the destination address);
<i>next_hop</i>	: the next hop;
<i>AS_path</i>	: ordered list of ASes traversed, with more-recently-visited ASes in front of less-recently-visited ASes;
<i>local_pref</i>	: local preference;
<i>med</i>	: multi-exit discriminator.

Each route  $r$  is associated with a 4-tuple  $rank(r)$ , defined as  $\langle r.local\_pref, \frac{1}{|r.AS\_path|}, \frac{1}{r.med}, \frac{1}{r.next\_hop} \rangle$ . For the destination  $d$ , all the routes available to a node  $i$  is ranked in lexical order by  $rank(\cdot)$ , and the route with the highest rank is selected as the route of  $i$ , denoted as  $i.aspath$ .

Given a route  $r$  available to a node  $i$ , attribute  $r.local\_pref$  is determined by the *route ranking policy* of  $i$ . For convenience, we call the ranking policy that assigns  $r.local\_pref$  to a constant value the *shortest-path-first policy* or the *SPF policy*, where a route with the shortest AS-path ranks the highest.

Besides route ranking policy, BGP uses import and export policies. The *import policy* of a node  $i$  defines the set of import neighbors of  $i$  whose routes are accepted by  $i$ ; the *export policy* of  $i$  defines the set of export neighbors of  $i$  to which  $i$  announces its route.

### 3 Protocol CPV

Our objective is to design a path-vector routing protocol that satisfies the following two requirements.

- It contains continuously occurring faults (whether or not anticipated) locally around where they occur, such that the distance to which the fault effects propagate is a function of their duration.
- Once faults stop occurring, it converges from an arbitrary state to a legitimate state within time depending on the number of nodes perturbed by the faults.

To this end, we first describe the issue of fault propagation in BGP, and we then present the detailed design of our protocol, CPV.

#### 3.1 Fault propagation in BGP

To see how faults propagate unboundedly in BGP, we consider the simple network shown in Figure 1, where

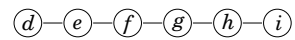


Figure 1: A simple line network. For simplicity, each node in the figure also represents its AS.

$d$  is the destination to which the other nodes maintain a route. Suppose  $d$  fail-stops at a state where all other nodes have learned their routes to  $d$ . Then  $e$  will withdraw its route to  $d$  and send a route-withdrawal message to  $f$ . Now suppose  $d$  rejoins the network before  $f$  withdraws its route. Then nodes  $f$ ,  $g$ ,  $h$ , and  $i$  should, ideally, not withdraw their routes anymore. Nevertheless, the interval between  $d$  fail-stopping and  $d$  rejoining and the fault history may well be such that BGP instability-suppression timers as well as route-flap-damping do not prevent nodes  $f$ ,  $g$ ,  $h$ , and  $i$  from withdrawing their routes (even though these nodes will learn later the same routes again). Thus, in this fault scenario, the faults propagate to all the nodes in the network, whereas the faults should ideally only affect  $e$ . Due to the fault propagation, the time taken for the network to stabilize after  $d$  joins is a function of the network size instead of the minimum number of nodes that should have been affected (which is 1 in this case).

To simplify presentation, we used a simple network and a simple fault scenario in the above discussion. In practice, networks are more connected where multiple paths exist for a destination, which can incur more network instability [20]; and there are more complex fault scenarios, with varying frequencies of fault occurrence and varieties of state corruptions [9, 11]. Therefore, the negative impact of fault propagation is even more severe in practice (as shown in Section 6 and in [19]). Thus, unbounded fault propagation in networks should be avoided.

#### 3.2 Protocol concepts

One reason why faults propagate unboundedly in the above example is as follows.

- When  $d$  rejoins the network, the route-withdrawal as the result of the fail-stop of  $d$  has already propagated to  $f$ .
- After  $d$  rejoins,  $e$  establishes its route to  $d$  and sends a route-announcement to  $f$ ; the route-announcement, however, lags behind the route-withdrawal (which reflects an obsolete state of the network) in the sense that  $f$  has sent out the route-withdrawal to  $g$  before  $f$  receives the new route-announcement from  $e$ .
- As a result, the route-withdrawal keeps propagating from  $f$  to  $g$ , and then to  $h$  and  $i$ , even if the route-announcement tails it to reach  $g$ ,  $h$ , and  $i$ .

**Design pattern for continuous containment.** To contain continuously occurring faults, we, therefore, need a mechanism that enables information regarding each new network state to catch up with and stop the propagation of information regarding the preceding state which has become obsolete. To this end, we design protocol CPV where the diffusing computation involved in path-vector routing consists of three diffusing waves propagating at different speed: *stabilization wave* at the lowest speed, *containment wave* at an intermediate speed, and *undo-containment wave* at the highest speed (see Figure 2). The three diffusing waves run in paral-

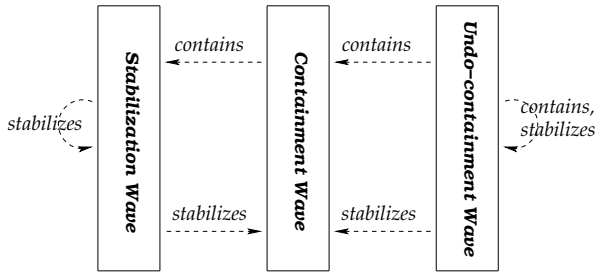


Figure 2: Parallel diffusing waves in CPV

lel and coordinate to contain the propagation of obsolete information while stabilizing a network at the same time:

- Whenever a node  $j$  needs to change its state, it engages a containment wave  $cw_0$  before engaging a new stabilization wave  $sw_1$ , so that  $cw_0$  stops the previous stabilization wave  $sw_0$  from propagating the existing state of  $j$  (which will become obsolete once  $j$  executes  $sw_1$ );
- In the presence of continuously occurring faults, another fault  $f$  may occur before  $j$  executes  $sw_1$ , then there are two cases:
  - (i)  $j$  does not need to change its state any more after  $f$  occurs. Then  $cw_0$  has to be stopped so that  $sw_0$  can keep propagating to nodes whose state still needs to be corrected. To this end,  $j$  engages an undo-containment wave  $uw_0$  which catches up with and stops  $cw_0$ .
  - (ii)  $j$  still needs to change its state after  $f$  occurs.

Then  $j$  does nothing but lets  $cw_0$  propagate (to stop  $sw_0$ ).

- Each stabilization as well as undo-containment wave stabilizes itself, and each containment wave is stabilized (and deactivated) by the corresponding stabilization or undo-containment wave.

We elaborate on the design pattern as follows.

**A. Containing a stabilization wave:** When a node needs to engage a new stabilization wave to change its state, the existing state of the node becomes obsolete but may have propagated to other nodes via the previous stabilization wave. To avoid unbounded propagation of its existing state, the node engages a containment wave before generating the new stabilization wave, and the containment wave will stop the previous stabilization wave. When a containment wave propagates from a node  $i$  to its neighbor  $j$ ,  $j$  further propagates the containment wave if  $j$  has propagated the obsolete information from  $i$ . For instance, in the example discussed in Section 3.1, when  $d$  rejoins,  $e$  will detect that it needs to change state (i.e., to establish a route to  $d$ ); therefore,  $e$  initiates a containment wave toward  $f$ , then  $f$  decides whether to propagate the containment wave depending on whether  $f$  has propagated the route-withdrawal to  $g$ , and so on.

To contain stabilization waves in the presence of continuously occurring faults, each containment wave from a node  $i$  carries the prediction of the state to which  $i$  will converge so that the neighbors of  $i$  are able to decide whether to hold a stabilization wave.<sup>2</sup> Moreover, to contain continuously occurring faults, the containment and stabilization waves that are related to the same fault should be able to co-exist. To this end, containment waves do not modify variables of stabilization waves, and each containment wave is a one-way open (instead of a two-way closed) diffusing process that is deactivated later by an associated stabilization or undo-containment wave.

**B. Containment-assisting stabilization wave:** Stabilization waves are initiated or propagated by nodes that need to change routes after the occurrence of faults. To enable continuous fault containment, stabilization waves adapt the basic path-vector routing algorithm with the following mechanisms.

- (i) When a node selects its route to a destination, it takes into account the predicted state of its neighbors.
- (ii) When a containment wave catches up with the corresponding stabilization wave at a node  $i$ , both the containment and the stabilization wave stop at  $i$  (this is enabled by not letting  $i$  join any stabilization wave). For instance, in the example discussed in Section 3.1, after  $d$  rejoins, if the contain-

<sup>2</sup>Note that the predicted state can be used to improve packet forwarding too, since it reflects a fresher network state.

ment wave initiated at  $e$  reaches  $f$  before  $f$  propagates the route-withdrawal,  $f$  will not propagate the route-withdrawal anymore, which prevents the route-withdrawal from propagating further.

On the other hand, to guarantee convergence (especially in the presence of transient loops in path-vector routing [14]), a node that has already joined a containment wave is free to join a stabilization wave without sacrificing continuous fault containment.

- (iii) Stabilization waves propagate slower than containment waves, so that containment waves can catch up with associated stabilization waves which are initiated earlier.

**C. Undoing a containment wave:** In the presence of continuously occurring faults, information that is obsolete at some point in time may become valid later when other faults occur. In this case, the containment wave marking that information as obsolete should be stopped and prevented from propagating further. This is enabled by letting the node that first detects this situation initiate an undo-containment wave which propagates along the same path as the corresponding containment wave. Since the undo-containment wave propagates faster than the containment wave does, the undo-containment wave is able to catch up with and stop the containment wave. Considering the example discussed in Section 3.1, for instance, when  $d$  fail-stops, a containment wave  $cw_0$  will propagate from  $e$  toward  $f$ , and so on; when  $d$  rejoins later,  $e$  will initiate another containment wave  $cw_1$  carrying the route-announcement that is “predicted” to take place later; suppose  $f$  has not withdrawn its route when it receives the predicted route-announcement from  $e$ ,  $f$  will detect that it should stop  $cw_0$ ; therefore,  $f$  initiates an undo-containment wave toward  $g$  to stop  $cw_0$ . (Note that  $f$  does not propagate  $cw_1$ , since  $f$  has not withdrawn its route.)

For the parallel-diffusing-wave based approach to work, each undo-containment wave must self-stabilize itself locally in the presence of faults (otherwise, we would need another type of diffusing wave to contain the undo-containment wave). This is achieved by ensuring that undo-containment waves use only those variables that are defined for stabilization and containment waves and no other variable.

A summary of the three-wave design is as follows.

- Containment waves contain the propagation of stabilization waves, and undo-containment waves contain the propagation of containment waves.
- Each containment wave is associated with two stabilization waves: the old stabilization wave which propagates obsolete information and which is contained, and the new stabilization wave which deactivates the containment wave after the latter has stopped the old stabilization wave.
- Each contained wave (e.g., a stabilization wave) sets

<b>Protocol</b>	<i>CPV.i</i>
<b>Constant</b>	$d$ : set of node-ids $d_s, d_c, d_u, I_{syn}$ : real
<b>Var</b>	$i.aspath, i.tp$ : list of AS-ids $i.j'.aspath, i.j'.tp$ ( $j' \in IM.i$ ) : list of AS-ids $i.ghost, i.j'.ghost$ ( $j' \in IM.i$ ) : boolean $i.t$ : real $k$ : node-id
<b>Parameter</b>	$j$ : node-id
<b>Action</b>	$\langle SW \rangle :: S.i.j \vee R.i \vee R'.i \xrightarrow{d_s}$ $\text{if } S.i.j \rightarrow i.aspath := aspath(i, j);$ $i.ghost := i.j.ghost$ $\square$ $R.i \rightarrow \text{if } i \in d \rightarrow i.aspath := [i.as]$ $\square$ $i \notin d \rightarrow i.aspath := \emptyset$ $\text{fi};$ $i.ghost := false$ $\text{fi};$ $i.tp := \emptyset;$ $i.t := CLK.i;$ $\text{send } m(i.aspath, i.ghost, i.tp) \text{ to } EX.i$ $\square$ $\langle CW \rangle :: (\neg i.ghost \wedge JC.i) \vee TP.i \xrightarrow{d_c}$ $i.ghost := true;$ $\text{do } S.i.k \rightarrow i.tp := aspath(i, k) \text{ od};$ $\text{do } S'.i.k \rightarrow i.tp := tp(i, k) \text{ od};$ $i.t := CLK.i;$ $\text{send } m(i.ghost, i.tp) \text{ to } EX.i$ $\square$ $\langle UW \rangle :: LC.i \vee CC.i \xrightarrow{d_u}$ $\text{if } LC.i \rightarrow i.ghost := false \text{ fi};$ $\text{if } \neg R'.i \rightarrow i.tp := \emptyset \text{ fi};$ $i.t := CLK.i;$ $\text{send } m(i.ghost, i.tp) \text{ to } EX.i$ $\square$ $***$ $\langle SYN_1 \rangle :: (i.t + I_{syn} \leq CLK.i) \vee (i.t > CLK.i) \rightarrow$ $i.t := CLK.i;$ $\text{send } m(i.aspath, i.ghost, i.tp) \text{ to } EX.i$ $\square$ $\langle SYN_2 \rangle :: \text{rcv } m \text{ from } j \rightarrow$ $\text{if } j \in IM.i \rightarrow$ $\text{update } i.j.aspath, i.j.ghost, \text{ and/or } i.j.tp$ $\text{fi}$

Figure 3: CPV: Fault-Containing Path Vector Routing

the boundary of its corresponding containing wave (e.g., a containment wave) such that the latter does not propagate beyond where the former has reached.

### 3.3 The design of CPV

The protocol CPV is shown in Figure 3, where the protocol constants, variables, and actions for each node  $i$  are defined.

**Constants.** CPV uses five constants:  $d, d_s, d_c, d_u$ , and  $I_{syn}$ .  $d$  denotes the ID of the destination to which all other nodes in the network need to maintain a route;  $d_s, d_c$ , and  $d_u$  are used to control the propagation speed of stabilization waves, containment waves, and undo-containment waves respectively;  $I_{syn}$  is used to control the frequency of information synchronization between neighboring nodes.

To contain continuously occurring faults,  $d_s$ ,  $d_c$ , and  $d_u$  should be such that  $d_s > \alpha \cdot (d_c + U)$ ,  $d_c > \alpha \cdot (d_u + U)$ , and  $d_u \geq 0$ . (Details of the derivation of these constants can be found in the proof of Theorem 1.)

**Variables.** As in BGP, node  $i$  maintains its AS-level path to  $d$ , denoted as  $i.aspath$ . To enable containment waves,  $i$  uses two additional variables:  $i.ghost$  and  $i.tp$ .  $i.ghost$  denotes whether or not  $i$  is involved in a containment wave, and  $i.tp$  denotes the predicted route which  $i$  will adopt next. To coordinate with its neighbors,  $i$  also maintains a copy of the three variables for each of its import neighbors  $j'$ , denoted as  $i.j'.aspath$ ,  $i.j'.ghost$ , and  $i.j'.tp$ .

For convenience, variable  $i.t$  is used to record the time when  $i$  sends a message to its export neighbors the last time; a dummy variable  $k$  is also used.

**Protocol actions.** CPV consists of five actions, one for each of the diffusing waves involved in CPV and two for updating information between neighbors. We briefly explain each action by its category. For convenience, we define the following notations:

$$\begin{aligned}
aspath(i, k) &: i.k.aspath, && \text{if } i.as = k.as \\
& [i.as, i.k.aspath], && \text{if } i.as \neq k.as; \\
tp(i, k) &: i.k.tp, && \text{if } i.as = k.as \\
& [i.as, i.k.tp], && \text{if } i.as \neq k.as; \\
rank(i, k) &: \max\{rank(i.k.aspath), rank(i.k.tp)\}; \\
N(i) &: \text{the next-hop of } i \text{ on its route to } d; \\
IM.i &: \text{the set of import neighbors of } i; \\
EX.i &: \text{the set of export neighbors of } i; \\
CLK.i &: \text{the current clock value of } i.
\end{aligned}$$

**Stabilization wave.** Stabilization waves are implemented by action  $SW$ .  $SW$  is executed when

- $i$  needs to propagate a stabilization wave from an import neighbor  $j$  (i.e.,  $S.i.j = true$ ),  $i$  needs to reset  $i.aspath$  (i.e.,  $R.i = true$ ), or  $i$  needs to reset  $i.tp$  (i.e.,  $R'.i = true$ ); and
- the above condition has continuously held for the past  $d_s$  time.

Corresponding to the intuitive formulation of stabilization waves in Section 3.2.B,

- $S.i.j \equiv$

$$\begin{aligned}
& i \notin d \wedge j \in IM.i \wedge i.j.aspath \neq \emptyset \wedge i.as \notin i.j.aspath \wedge \\
& (i.N(i).aspath \neq \emptyset \wedge i.as \notin i.aspath \wedge \neg i.ghost \Rightarrow \\
& \quad \neg i.N(i).ghost) \wedge \\
& ((\neg i.j.ghost \wedge (\forall k : k \in IM.i \wedge k \neq j \Rightarrow \\
& \quad rank(i, k) < rank(i.j.aspath))) \vee \\
& (i.j.ghost \wedge (\forall k : k \in IM.i \wedge k \neq j \Rightarrow \\
& \quad (i.k.aspath \neq \emptyset \Rightarrow i.k.ghost) \wedge \\
& \quad rank(i, k) < rank(i.j.aspath)))) \\
& ) \wedge \\
& (i.as \in i.aspath \vee (j = N(i) \wedge i.aspath \neq aspath(i, j)) \vee \\
& (j \neq N(i) \wedge rank(i.N(i).aspath) < rank(i.j.aspath)))
\end{aligned}$$

- $R.i \equiv$

$$\begin{aligned}
& (i \in d \wedge i.aspath \neq [d]) \vee \\
& (i \notin d \wedge i.aspath \neq \emptyset \wedge \\
& (\forall k : k \in IM.i \Rightarrow (i.k.tp = \emptyset \vee i.as \in i.k.tp) \wedge \\
& (i.k.aspath = \emptyset \vee i.as \in i.k.aspath)))
\end{aligned}$$

- $R'.i \equiv$

$$\begin{aligned}
& i.tp \neq \emptyset \wedge \\
& (\forall k : k \in IM.i \Rightarrow (i.k.tp = \emptyset \vee i.as \in i.k.tp) \wedge \\
& (i.k.aspath = \emptyset \vee i.as \in i.k.aspath))
\end{aligned}$$

To execute  $SW$ ,  $i$  updates  $i.aspath$  and  $i.ghost$  appropriately. Since the execution of  $SW$  means fixing the route to a destination,  $i$  always reset  $i.tp$  to empty, signifying that  $i$  will not change route any more unless faults occur again. After updating its state,  $i$  sends the updated state to its export neighbors.

**Containment wave.** Containment waves are implemented by action  $CW$ .  $CW$  is executed when

- $i$  needs to join a containment wave (i.e.,  $\neg i.ghost \wedge JC.i$  holds), or  $i$  needs to update  $i.tp$  (i.e.,  $TP.i = true$ ); and
- the above condition has continuously held for the past  $d_c$  time.

Corresponding to the intuitive formulation of containment waves in Section 3.2.A,

- $JC.i \equiv$

$$\begin{aligned}
& (i \in d \wedge i.aspath \neq [d]) \vee \\
& (i \notin d \wedge ((\exists k : S.i.k) \vee R.i \vee \\
& (i.N(i).ghost \wedge i.aspath = aspath(i, N(i))))
\end{aligned}$$

- $TP.i \equiv$

$$\begin{aligned}
& (\exists k : S.i.k \wedge i.tp \neq aspath(i, k)) \vee \\
& ((i.ghost \vee JC.i) \wedge \\
& (\exists k : S'.i.k \wedge i.tp \neq tp(i, k) \wedge i.aspath \neq tp(i, k)))
\end{aligned}$$

where  $S'.i.j \equiv$

$$\begin{aligned}
& i \notin d \wedge j \in IM.i \wedge i.j.tp \neq \emptyset \wedge i.as \notin i.j.tp \wedge \\
& (\forall k : k \in IM.i \wedge k \neq j \Rightarrow rank(i, k) < rank(i.j.tp))
\end{aligned}$$

To execute  $CW$ ,  $i$  sets  $i.ghost$  as  $true$  to signify that it is involved in a containment wave and will change route soon.  $i$  also updates  $i.tp$  as appropriate.

[Improved packet forwarding via CW] Since containment waves carry information on relatively fresher network state, containment waves help in forwarding packets in the presence of faults. For example, when  $i$  is the first node in a containment wave that has a non-empty predicted route  $i.tp$ ,<sup>3</sup>  $i$  can use  $i.tp$  instead of  $i.aspath$  to forward packets, since  $i.tp$  is a better route than  $i.aspath$ . Similarly, when  $i$  is involved in a containment wave but has no alternative route to the destination,<sup>4</sup>  $i$  can immediately stop forwarding packets destined for the destination to avoid wasting network resources.

**Undo-containment wave.** Undo-containment waves are implemented by action  $UW$ .  $UW$  is executed when

- $i$  is involved in a containment wave but does not need to change route any more (i.e.,  $LC.i = true$ ), or the part of the state of  $i$  related to containment waves has been corrupted (i.e.,  $CC.i = true$ ); and

<sup>3</sup>That is,  $(i.ghost \wedge i.tp \neq \emptyset \wedge \neg i.N(i, i.tp).ghost)$  holds, where  $N(i, i.tp)$  denotes the next-hop of  $i$  on route  $i.tp$ .

<sup>4</sup>That is,  $i.ghost = true$  and  $i.tp = \emptyset$ .

- the above condition has continuously held for the past  $d_u$  time.

Corresponding to the intuitive formulation of undo-containment waves in Section 3.2.C,

- $LC.i \equiv$

$$\begin{aligned} & i.ghost \wedge \\ & ((\exists k : S'.i.k \wedge i.aspath = tp(i, k)) \vee \\ & (\neg JC.i \wedge \neg(\exists k : S'.i.k)) \vee (R'.i \wedge \neg R.i)) \end{aligned}$$

- $CC.i \equiv$

$$\begin{aligned} & (i.tp \neq \emptyset \wedge \neg(\exists k : S.i.k \vee S'.i.k)) \vee \\ & (\neg i.ghost \wedge i.tp \neq \emptyset \wedge \neg R'.i) \vee i.as \in i.tp \end{aligned}$$

To execute  $UW$ ,  $i$  resets  $i.ghost$  to *false*, and if there is still a route to the destination (i.e.,  $R'.i = \text{false}$ ),  $i$  resets  $i.tp$  to empty.

Note that, when  $d$  keeps fail-stopping and rejoining at high frequencies, it is critical to the fault containment that the reset of  $i.tp$  be performed at the speed of stabilization waves instead of that of undo-containment waves; otherwise, the reset of  $i.aspath$  could potentially propagate far away.

**Information update.** Actions  $SYN_1$  and  $SYN_2$  enable neighboring nodes to exchange information so that information consistency is guaranteed.

- Action  $SYN_1$ : if  $i$  has not updated its state with its export neighbors for more than  $I_{syn}$  time (i.e.,  $t.i + I_{syn} \leq CLK.i$ ) or if  $i.t$  is corrupted to be greater than the current clock value,  $i$  sets  $i.t$  to its current clock value, and sends its state (i.e.,  $i.aspath$ ,  $i.ghost$ , and  $i.tp$ ) to its export neighbors.
- Action  $SYN_2$ : when  $i$  receives a state update from an import neighbor  $j$ ,  $i$  updates its state record regarding  $j$ .

### 3.4 Example revisited

We reconsider the example discussed in Section 3.1 by examining how the network behaves if CPV is adopted. For simplicity of presentation, we assume that  $\alpha = 1$ , link delay is a constant  $u$ , processing delay is negligible, and the propagation speed of undo-containment waves is twice that of containment waves, which in turn is twice that of stabilization waves (i.e.,  $d_s = 2d_c + u$  and  $d_e = 2d_u + u$ ).

When  $d$  fail-stops, actions  $SW$  and  $CW$  are enabled at  $e$  since both of the predicates  $R.e$  and  $JC.e$  hold. Given that  $d_c < d_s$ ,  $e$  initiates a containment wave  $cw_1$  (by executing action  $CW$ ) earlier than it initiates a stabilization wave  $sw_1$ . Now suppose  $sw_1$  has reached  $f$  (i.e., action  $SW$  is enabled at  $f$ ) and  $cw_1$  has reached  $g$  (i.e., action  $CW$  is enabled at  $g$ ) when  $d$  rejoins. After  $d$  rejoins, actions  $SW$  and  $CW$  are enabled at  $e$  again since both of the predicates  $S.e.d$  and  $JC.e$  hold. After  $d_c + u$  time, the second containment wave  $cw_2$  from  $e$  reaches  $f$ , carrying the predicted state of  $e$  (i.e.,  $e.tp$  as  $[d.as]$ ); at the same time,  $cw_1$  reaches  $h$ . When  $cw_2$  reaches

$f$ , action  $SW$  becomes disabled at  $f$  (since  $R.i$  becomes *false*), thus  $sw_1$  stops at  $f$ ; at the same time, action  $UW$  becomes enabled at  $f$  (since  $LC.f$  holds), while  $CW$  remains disabled at  $e$  (which means that  $cw_2$  stops at  $f$ ). Since undo-containment waves propagate twice as fast as containment waves, the undo-containment wave  $uw_1$  initiated from  $f$  will catch up with  $cw_1$  at  $h$ , after which action  $CW$  becomes disabled at  $h$  and  $cw_1$  as well as  $uw_1$  stops at  $h$ .

In the above scenario, therefore, stabilization waves propagate to  $f$  the farthest, containment as well as undo-containment waves propagate to  $h$  the farthest, and node  $i$  is not affected. Moreover, traffic forwarding is only slightly affected in the sense that only  $e$  changes its route in the presence of faults and that  $e$  recovers its route (via  $e.tp$ ) as soon as  $d$  rejoins.

## 4 Continuous containment and local stabilization

Toward establishing the foundation for studying and toward precisely characterizing system properties in the presence of continuously occurring faults, we formulate notions related to fault containment and local stabilization: perturbed node, contaminated node, perturbation size, contamination range,  $\mathcal{F}$ -containment, and  $\mathcal{F}$ -stabilization.

We regard a *system* as the union of a network and the protocols running on it. In the presence of faults, a network  $G$  may change in the sense that its topology or routing policy function changes, where the topology of  $G$  is the subgraph  $G'(V', E')$  of  $G(V, E)$  such that  $V' = \{i : i \in V \wedge i \text{ is up}\}$  and  $E' = \{(i, j) : i \in V' \wedge j \in V' \wedge (i, j) \in E \wedge (i, j) \text{ is up}\}$ . To reflect changes in network topology and routing policy function, we regard the state of a system as the union of the network topology, the routing policy function, and the state of all the up nodes, with the state of a node being the values of the variables maintained at the node. At a system state  $q$ , the network topology is denoted by  $G.q(V.q, E.q)$ , and the state of a node  $j$  is denoted as  $j.q$ .

We characterize the behavior of a system in the presence of faults by the *system history*. A *system history*  $\mathcal{H}$  is either a finite sequence  $q_0, (e_1, t_1), q_1, (e_2, t_2), \dots, q_n$ , or an infinite sequence  $q_0, (e_1, t_1), q_1, (e_2, t_2), \dots, q_{k-1}, (e_k, t_k), q_k, \dots$ , of alternating system states (i.e.  $q_0, q_1, \dots$ ) and events (i.e.  $e_1, e_2, \dots$ ), where

- An event is either the execution of a protocol action or the occurrence of a fault;
- For every  $k \geq 1$ ,  $t_k \leq t_{k+1}$ , and each state transition  $q_{k-1}, (e_k, t_k), q_k$  means that  $e_k$  at time  $t_k$  changes the system state from  $q_{k-1}$  to  $q_k$ ; and
- For any two pairs  $(e_k, t_k)$  and  $(e_{k'}, t_{k'})$  in  $\mathcal{H}$  ( $k \neq k'$ ), if  $e_k$  and  $e_{k'}$  occur at the same node, then  $t_k \neq t_{k'}$  (i.e., at most one event can occur at a node at any

time).

$\mathcal{H}$  is a finite sequence only if it ends with a state  $q_n$  such that there is no action enabled at  $q_n$  and no fault occurs after the system reaches  $q_n$ .

A subsequence  $\alpha$  of a system history  $\mathcal{H}$  is called a *history segment* if  $\alpha$  starts and ends with a state. A history segment  $\beta$  is called a *history prefix* if  $\beta$  starts with the initial state  $q_0$ . For convenience, we denote as  $\mathcal{H}(q)$  a history prefix ending with a state  $q$ . A history segment  $\gamma$  starting at a state  $q_k$  is called a *computation starting at  $q_k$*  if  $\gamma$  is the suffix of  $\mathcal{H}$  starting at  $q_k$  and every event in  $\gamma$  is the execution of a protocol action (i.e., no fault occurs in  $\gamma$ ).

Given a network, a protocol specification defines a set of legitimate states. For example, given a network topology, a routing policy function, and a destination, the specification of BGP determines a set of legitimate states, each of which specifies the route of every node. For a self-stabilizing protocol, each computation  $C_k$  starting at an arbitrary state  $q_k$  determined a converged state  $L(q_k, C_k)$ . Whenever a fault changes a system state from  $q_k$  to  $q_{k+1}$ , the instance of computation changes in the sense that the computation  $C_{k+1}$  starting at  $q_{k+1}$  is different from  $C_k$ . Then, given a state  $q_k$  with a history prefix  $\mathcal{H}(q_k)$ , we regard as a *protocol execution  $\mathcal{E}(q_k)$*  a set of computations each of which specifies a computation  $C(q_{k'}, \mathcal{E}(q_k))$  for a different state  $q_{k'}$  in  $\mathcal{H}(q_k)$  that is either the initial state or a state reached immediately after a fault occurs. Then, given an arbitrary state  $q_k$  and an arbitrary protocol execution  $\mathcal{E}(q_k)$ , we define the *stabilization-set of  $q_k$  under  $\mathcal{E}(q_k)$* , denoted by  $S_s(q_k, \mathcal{E}(q_k))$ , as the set of nodes that need to change state in order for the network to stabilize from  $q_k$ , i.e.,  $S_s(q_k, \mathcal{E}(q_k)) = \{j : j \in V.q_k \wedge j.q_k \neq j.L(q_k, C(q_k, \mathcal{E}(q_k)))\}$ . (Of course, if  $q_k$  is a legitimate state,  $S_s(q_k, \mathcal{E}(q_k)) = \emptyset$ .)

If an event  $e$  occurring at time  $t_k$  changes the system state from  $q_{k-1}$  to  $q_k$  under a protocol execution  $\mathcal{E}(q_k)$ , we define the *corruption set of  $e$  at  $t_k$* , denoted by  $cpt(e, t_k, \mathcal{E}(q_k))$ , as the set of nodes that need not change state in order for the network to stabilize from  $q_{k-1}$ , but need to change state in order for the network to stabilize from  $q_k$ , i.e.,  $cpt(e, t_k, \mathcal{E}(q_k)) = S_s(q_k, \mathcal{E}(q_k)) \setminus S_s(q_{k-1}, \mathcal{E}(q_k))$ . In addition, if  $e$  is not a state corruption, we define the *correction set of  $e$  at  $t_k$* , denoted by  $cct(e, t_k, \mathcal{E}(q_k))$ , as the set of nodes in  $V.q_k$  that need to change state in order for the network to stabilize from  $q_{k-1}$ , but need not change state in order for the network to stabilize from  $q_k$ , i.e.,  $cct(e, t_k, \mathcal{E}(q_k)) = (S_s(q_{k-1}, \mathcal{E}(q_k)) \setminus S_s(q_k, \mathcal{E}(q_k))) \cap V.q_k$ . If  $e$  is a state corruption, we define  $cct(e, t_k, \mathcal{E}(q_k))$  as  $\emptyset$ .

**Perturbation vs. contamination.** For every node  $j \in cpt(e, t_k, \mathcal{E}(q_k))$ , we regard  $j$  as *perturbed* by  $e$  if  $e$  is a fault, and we regard  $j$  as *contaminated* via  $e$  if  $e$  is the execution of a protocol action; for every node  $j \in cct(e, t_k, \mathcal{E}(q_k))$ , we regard  $j$  as *corrected* by  $e$ . For

instance, in the example discussed in Section 3.1, when  $d$  fail-stops,  $e, \dots, i$  are perturbed by the fail-stop of  $d$ ; when  $d$  rejoins before  $f$  withdraws its route,  $f, \dots, i$  are corrected by the join of  $d$ ; when BGP is used,  $f, \dots, i$  continue to withdraw their routes after  $d$  rejoins, thus  $f, \dots, i$  are contaminated via BGP actions.

Given a system state  $q_k$  with a history prefix  $\mathcal{H}(q_k)$  and a protocol execution  $\mathcal{E}(q_k)$ , a node  $i$  is *perturbed* at  $q_k$  if either of the following conditions hold:

- $q_k$  is the initial state (i.e.,  $k = 0$ ), and  $i$  needs to change state in order for the network to stabilize.
- $i$  has been perturbed by a fault at some point, and neither  $i$  has been corrected by a fault nor has the network reached a legitimate state ever since.

A node  $i$  is *contaminated* at  $q_k$  if  $i$  has been contaminated at some point and has not been corrected ever since.

Formally,

**Definition 1 (Perturbed node)** *Given a protocol execution  $\mathcal{E}(q_k)$ , and a network state  $q_k$  with history prefix  $\mathcal{H}(q_k)$  ( $k \geq 0$ ), a node  $i$  ( $i \in V.q_k$ ) is perturbed at  $q_k$  if and only if*

- $i \in S_s(q_k, \mathcal{E}(q_k))$ ,  
when  $k = 0$
- $PT(i, q_0, q_k, \mathcal{H}(q_k), \mathcal{E}(q_k)) = true$ ,  
when  $k > 0 \wedge \neg(\exists k' : q_{k'} \in \mathcal{H}(q_k) \wedge q_{k'} \in Q_L(q_{k'}))$
- $PT(i, LL(q_k, \mathcal{H}(q_k)), q_k, \mathcal{H}(q_k), \mathcal{E}(q_k)) = true$ ,  
when  $k > 0 \wedge (\exists k' : q_{k'} \in \mathcal{H}(q_k) \wedge q_{k'} \in Q_L(q_{k'}))$

where

$$\begin{aligned}
PT(i, q_m, q_n, \mathcal{H}(q_n), \mathcal{E}(q_k)) &\equiv \\
& q_m \in \mathcal{H}(q_n) \wedge \\
& ((e_n \text{ is a fault} \wedge i \in cpt(e_n, t_n, \mathcal{E}(q_k))) \vee \\
& (\exists k' : m \leq k' < n \wedge i \text{ is perturbed at } q_{k'} \wedge \\
& \neg(\exists j' : k' < j' \leq n \wedge (e_{j'}, t_{j'}) \in \mathcal{H}(q_n) \wedge \\
& e_{j'} \text{ is a fault} \wedge i \in cct(e_{j'}, t_{j'}))) \\
LL(q_k, \mathcal{H}(q_k)) &= q_j \text{ such that} \\
& q_j \in \mathcal{H}(q_k) \wedge q_j \in Q_L(q_j) \wedge \\
& \neg(\exists j' : j < j' \leq k \wedge q_{j'} \in \mathcal{H}(q_k) \wedge q_{j'} \in Q_L(q_{j'})) \\
Q_L(q_j) &= \text{the set of legitimate states corresponding} \\
& \text{to the network topology and routing policy} \\
& \text{function at } q_j.
\end{aligned}$$

**Definition 2 (Contaminated node)** *Given a protocol execution  $\mathcal{E}(q_k)$ , and a network state  $q_k$  with history prefix  $\mathcal{H}(q_k)$  ( $k \geq 0$ ), a node  $i$  ( $i \in V.q_k$ ) is contaminated at  $q_k$  if and only if*

- $CT(i, q_0, q_k, \mathcal{H}(q_k), \mathcal{E}(q_k)) = true$ ,  
when  $k > 0 \wedge \neg(\exists k' : q_{k'} \in \mathcal{H}(q_k) \wedge q_{k'} \in Q_L(q_{k'}))$
- $CT(i, LL(q_k, \mathcal{H}(q_k)), q_k, \mathcal{H}(q_k), \mathcal{E}(q_k)) = true$ ,  
when  $k > 0 \wedge (\exists k' : q_{k'} \in \mathcal{H}(q_k) \wedge q_{k'} \in Q_L(q_{k'}))$

where

$$\begin{aligned}
CT(i, q_m, q_n, \mathcal{H}(q_n), \mathcal{E}(q_k)) &\equiv \\
& q_m \in \mathcal{H}(q_n) \wedge \\
& ((e_n \text{ is not a fault} \wedge i \in cpt(e_n, t_n, \mathcal{E}(q_k))) \vee \\
& (\exists k' : m \leq k' < n \wedge i \text{ is contaminated at } q_{k'} \wedge \\
& \neg(\exists j' : k' < j' \leq n \wedge (e_{j'}, t_{j'}) \in \mathcal{H}(q_n) \wedge \\
& i \in cct(e_{j'}, t_{j'})))
\end{aligned}$$



$LL(q_k, \mathcal{H}(q_k)) = q_j$  such that  
 $q_j \in \mathcal{H}(q_k) \wedge q_j \in Q_L(q_j) \wedge$   
 $\neg(\exists j' : j < j' \leq k \wedge q_{j'} \in \mathcal{H}(q_k) \wedge q_{j'} \in Q_L(q_{j'}))$   
 $Q_L(q_j) =$  the set of legitimate states corresponding  
to the network topology and routing policy  
function at  $q_j$ .

Intuitively, these definitions imply that, a perturbed node remains perturbed until it is corrected by a fault or the network reaches a legitimate state; a contaminated node remains contaminated until it is corrected by either a fault or the execution of a protocol action. For instance, in the example discussed in Section 3.1, at the state immediately after  $d$  fail-stops,  $e, \dots, i$  are all perturbed; at the state immediately after  $d$  rejoins, only  $e$  is perturbed, since  $f, \dots, i$  are corrected by the rejoining of  $d$ ; at the state where  $f$  withdraws its route,  $f$  is contaminated.

The number of perturbed nodes at a system state, which we define as the *perturbation size*, reflects the severity of the impact that faults have on the system. Formally,

**Definition 3 (Perturbation size)** Given a system state  $q_k$  with a history prefix  $\mathcal{H}(q_k)$  and a protocol execution  $\mathcal{E}(q_k)$ , the perturbation size at  $q_k$ , denoted by  $\mathcal{P}(q_k, \mathcal{H}(q_k), \mathcal{E}(q_k))$ , is the number of nodes that are perturbed at  $q_k$ . That is,  $\mathcal{P}(q_k, \mathcal{H}(q_k), \mathcal{E}(q_k)) = |\{i : i \in V.q_k \wedge i \text{ is perturbed at } q_k\}|$ .

A set  $S$  of nodes are *contiguous* at a system state  $q$  if  $S \subseteq V.q$  and the subgraph of  $G.q(V.q, E.q)$  on  $S$  is connected, i.e., the graph  $G'(V', E')$  is connected, where  $V' = S$  and  $E' = \{(i, j) : i \in S \wedge j \in S \wedge (i, j) \in E.q\}$ . We regard a maximal set of perturbed nodes that are contiguous as a *perturbed region*. We also regard a contaminated node  $i$  as being *contaminated by a perturbed region*, via the execution of a protocol action  $a$ , if  $a$  is executed at  $i$  because of the state changes at a node  $j$  that is either in or contaminated by the perturbed region. Then, given a perturbed region  $S_p$  at a state  $q_k$ , the maximum distance from the nodes contaminated by  $S_p$  to  $S_p$ , which we define as the *contamination range* of  $S_p$  at  $q_k$ , reflects the severity of fault propagation from  $S_p$  at  $q_k$ . Formally,

**Definition 4 (Contamination range)** Given a perturbed region  $S_p$  at a state  $q_k$ , the contamination range of  $S_p$  at  $q_k$ , denoted by  $R(S_p, q_k)$ , is

$$\max_{i \in S_c} \text{hops}(i, S_p, q_k)$$

where

$$\begin{aligned}
S_c &= \{i : i \in V.q_k \wedge \\
&\quad i \text{ is contaminated by } S_p\}, \\
\text{hops}(i, S_p, q_k) &= \min_{j \in S_p} \text{hops}(i, j, q_k), \\
\text{hops}(i, j, q_k) &= \text{the number of hops in a shortest} \\
&\quad \text{path between } i \text{ and } j \text{ in } G.q_k.
\end{aligned}$$

To prevent fault propagation and to increase the stability as well as the availability of networks in the pres-

ence of continuously occurring faults, it is desirable that, at every state, the contamination range of each perturbed region remain bounded relative to the size of the perturbed region. Formally, this property is characterized as

**Definition 5 ( $\mathcal{F}$ -containment)** A system is  $\mathcal{F}$ -containing if and only if

For every perturbed region  $S_p$  at an arbitrary system state  $q_k$  ( $k \geq 0$ ),  $R(S_p, q_k) = O(\mathcal{F}(|S_p|))$ , where  $\mathcal{F}$  is a function.

Besides containing faults locally around where they occur, it is desirable that, once faults stop occurring (either indefinitely or for a long enough period), the network stabilizes quickly and, intuitively, within time depending on the perturbation size. Formally, this property is characterized as

**Definition 6 ( $\mathcal{F}$ -stabilization)** A system is  $\mathcal{F}$ -stabilizing if and only if

Starting at an arbitrary state  $q_k$  with an arbitrary history prefix  $\mathcal{H}(q_k)$  and an arbitrary protocol execution  $\mathcal{E}(q_k)$ , the system computation is guaranteed to reach a legitimate state within  $O(\mathcal{F}(\mathcal{P}(q_k, \mathcal{H}(q_k), \mathcal{E}(q_k))))$  time in the absence of faults, where  $\mathcal{F}$  is a function.

## 5 Analysis of CPV

Given a network topology  $G'(V', E')$  and a routing policy function  $P'$ , we let

$$\mathcal{L} \equiv (\forall i : i \in V' \Rightarrow \neg i.\text{ghost} \wedge i.\text{tp} = \emptyset \wedge LH.i)$$

where  $LH.i \equiv$

$$\begin{aligned}
&(i \in d \Rightarrow i.\text{aspath} = \emptyset) \wedge \\
&(i \notin d \Rightarrow ((d \subseteq V' \Rightarrow i.\text{aspath} = \text{aspath}(i, hr(i, P'))) \wedge \\
&\quad (d \not\subseteq V' \Rightarrow i.\text{aspath} = \emptyset))) \\
&\text{with } hr(i, P') \text{ being the highest-ranked neighbor of } i, \text{ i.e.} \\
&(\forall k : k \in IM.i \wedge k \neq hr(i, P') \Rightarrow \text{rank}(i.k.\text{aspath}) < \\
&\quad \text{rank}(i.hr(i, P').\text{aspath}))
\end{aligned}$$

Then, every state in  $\mathcal{L}$  is a state where every up node in the network has found its best route to the destination. Thus every state in  $\mathcal{L}$  is a legitimate state.

To analyze the properties of CPV, we first prove that CPV is self-stabilizing, then we prove that CPV contains continuously occurring faults and locally stabilizes.

**Lemma 1 (Self-stabilization)** Starting at an arbitrary state, every computation of a system where CPV is used is guaranteed to reach a state in  $\mathcal{L}$ .

**Proof:** To prove this lemma, we first prove the stabilization property for the common case where the SPF policy is used; then we prove the case where general route ranking policy (which may not be the SPF policy) is used (by the help of existing results on the convergence properties of Simple Path Vector Protocols [?]).

The SPF policy.

First, we introduce the concept of “computation rounds” which will be used in the proof. A system computation  $\mathcal{H}$  can be regarded as a sequence of *rounds*. A round is a minimal computation segment  $\gamma$  that starts at a state  $q_k$  ( $k \geq 0$ ) and, in  $\gamma$ , (i) every up node that has an action  $a$  continuously enabled from some time  $t'$  to  $(t' + d.a)$ , where  $t' \leq t_k$  and  $(t' + d.a) \geq t_k$ , executes at least one action, and (ii) if a message is sent to a node  $i$ , the action that receives the message must be executed at  $i$ . (We assume  $t_0$  is the time when  $\mathcal{H}$  starts.) For CPV, each round of a computation lasts at most  $(d_s + U)$  time in length.

When the SPF policy is used, we prove that  
**[Claim 0]** Starting at an arbitrary state, every computation of a system where CPV and the SPF policy are used is guaranteed to reach a state in  $\mathcal{L}$  within  $O(|V'|)$  time.

To prove Claim 0, we analyze the two different cases that can exist: when  $d$  is up (i.e.,  $d \subseteq V'$ ), and when  $d$  is down (i.e.,  $d \not\subseteq V'$ ).

**[Claim 0.0]** Starting at an arbitrary state when  $d$  is up (i.e.,  $d \subseteq V'$ ), every computation of a system where CPV and the SPF policy are used is guaranteed to reach a state in  $\mathcal{L}$  within  $O(D)$  time, where  $D$  is the number of hops in the longest shortest path from a node not in  $d$  to  $d$  in  $G'$ .

In this case, we prove the claim by the “convergence stair” approach used in [2] and some other works. That is, we exhibit a finite sequence of state predicates  $\mathcal{L}.0, \mathcal{L}.1, \dots, \mathcal{L}.D$  such that

- (i) Starting at an arbitrary state, the system is guaranteed to reach a state in  $\mathcal{L}.0$ .
- (ii) For each  $l$  such that  $0 \leq l \leq D$ :  
 $\mathcal{L}.l$  is closed under system execution; that is, once  $\mathcal{L}.l$  holds in an arbitrary system computation, it continues to hold subsequently.
- (iii) For each  $l$  such that  $0 \leq l < D$ :  
 Upon starting at an arbitrary state in  $\mathcal{L}.l$ , the system is guaranteed to reach a state in  $\mathcal{L}.(l + 1)$  within a constant amount of time.
- (iv)  $\mathcal{L}.D \equiv \mathcal{L}$

For our protocol,  $\mathcal{L}.0, \mathcal{L}.1, \dots, \mathcal{L}.D$  are defined as follows,

- $\mathcal{L}.0 \equiv (\forall i : i \in d \Rightarrow i.aspath = \emptyset)$
- For  $0 \leq l < D$ :

$$\mathcal{L}.(l + 1) \equiv \mathcal{L}.l \wedge (\forall i : i \in V' \wedge hops(i, d, G') = l + 1 \Rightarrow \neg i.ghost \wedge i.tp = \emptyset \wedge LH.i)$$

where  
 $hops(i, d, G')$  = the number of hops in the shortest path from  $i$  to  $d$  in  $G'$ .

We prove the stabilization of  $\mathcal{L}$  by proving the individual elements of the “convergence stair” method:

- (i) Starting at an arbitrary state  $q_0$ , the system is guaranteed to reach a state in  $\mathcal{L}.0$  within constant (i.e.,  $d_s$ ) time.

If  $\mathcal{L}.0$  holds at state  $q_0$ , we are done.

If  $\mathcal{L}.0$  does not hold at  $q_0$ , then  $R.i$  must hold for some node  $i \in d$ . For each of such node  $i$ , therefore, action  $SW$  is enabled at  $q_0$  and will be executed within  $d_s$  time.

Therefore, starting at an arbitrary state  $q_0$ , the system is guaranteed to reach a state in  $\mathcal{L}.0$  within  $d_s$  time.

- (ii) For each  $l$  such that  $0 \leq l \leq D$ :  $\mathcal{L}.l$  is closed under system execution.

To prove this claim, we first prove that, starting at state  $q_0$ , all invalid routes that are of hop-length  $l'$  are removed within  $l'$  rounds of system computation. This comes from the fact that, for every such invalid route  $r$ , nodes in the ASes of  $r$  execute action  $SW$  which removes subpath  $r$  gradually. Therefore, this sub-claim is easily proved by induction on the number of hops from an AS in  $r$  to  $d$ . (For conciseness, we skip the detail here.)

Therefore, once the system reaches a state in  $\mathcal{L}.l$ , neither action  $SW$  nor action  $CW$  will be enabled for nodes whose minimum AS-level hop distance to  $d$  is less than or equal to  $l$  (i.e.,  $hops(i, d, G') \leq l$  for each of such node  $i$ ), as a result of which no such node will change state any more. Thus, once the system reaches a state in  $\mathcal{L}.l$ , the system state remains in  $\mathcal{L}.l$  subsequently.

- (iii) For each  $l$  such that  $0 \leq l < D$ : Upon starting at an arbitrary state in  $\mathcal{L}.l$ , the system is guaranteed to reach a state in  $\mathcal{L}.(l + 1)$  within  $(d_s + U)$  time.

To prove this, we only need to prove that for a node  $i$  where  $hops(i, d, G') = l + 1$  but  $\neg i.ghost \wedge i.tp \wedge LH.i$  does not hold when the system is at state  $\mathcal{L}.l$ ,  $\neg i.ghost \wedge i.tp \wedge LH.i$  will become holding within  $(d_s + U)$  time. This is true because, if  $\neg i.ghost \wedge i.tp \wedge LH.i$  does not hold when the system is at state  $\mathcal{L}.l$ ,  $S.i.k = true$  for some import neighbor  $k$  of  $i$  where  $hops(k, d, G') = l$ ; thus, action  $SW$  is enabled at  $i$  and will be executed within  $d_s$  time; given that it takes up to  $U$  time for the new state of  $k$  to reach  $i$ ,  $i$  executes action  $SW$  within  $d_s + U$  time after the system reaches a state in  $\mathcal{L}.l$ , by which  $LH.i$  becomes *true*,  $i.ghost$  is set to *false*, and  $i.tp$  is set to  $\emptyset$ .

- (iv)  $\mathcal{L}.D \equiv \mathcal{L}$

This trivially holds since the maximum minimum-hop-distance from a node in  $V'$  to  $d$  is  $D$ .

Therefore, Starting at an arbitrary state when  $d$  is up (i.e.,  $d \subseteq V'$ ), every computation of a system where CPV and the SPF policy are used is guaranteed to reach a state in  $\mathcal{L}$  within  $O(D)$  time.

**[Claim 0.1]** Starting at an arbitrary state  $q_0$  when  $d$  is done (i.e.,  $d \not\subseteq V'$ ), every computation of a system where CPV and the SPF policy are used is guaranteed to reach a state in  $\mathcal{L}$  within  $O(|V'|)$  time.

Claim 0.1 is easily proved by following the same ap-

proach as used in proving Claim 0.0 and in [?]. That is, by considering each route  $r$  that exists at state  $q_0$ , then nodes in the ASes of  $r$  execute action  $SW$  which removes subpath of  $r$  gradually. Therefore, by induction on the number of hops from an AS in  $r$  to  $d$ , we see that, starting at  $q_0$ , there will be no route in the system within  $O(|V'|)$  rounds of computation, since the maximum hop-length of a loop-free route is  $|V'|$ , and every looping route is removed within constant time by executing action  $SW$ . Therefore, starting at state  $q_0$ , the system will reach a state where no node has a route to  $d$  (i.e., a state in  $\mathcal{L}$ ) within  $O(|V'|)$  time.

Having proved the lemma for the case where the SPF policy is used, we prove the lemma as follows for cases where non-SPF policies are used.

#### Non-SPF policies.

To prove the convergence of CPV from an arbitrary state to a state in  $\mathcal{L}$  when route ranking policies other than the SPF policy is used, we first prove that [Claim 1] starting at an arbitrary state  $q_k$ , the system reaches a state, within constant time, where the state related to containment waves and stabilization waves is consistent with each other, that is, a state in  $\mathcal{L}_0$ , with  $\mathcal{L}_0$  defined as  $(\forall i : CNST0.i \wedge CNST1.i)$ , where  $CNST0.i$  is defined as

$$i.tp \neq \emptyset \Rightarrow (i.ghost \wedge (\neg TP.i \Rightarrow (\exists k : (S.i.k \wedge i.tp = aspath(i, k)) \vee (S'.i.k : i.tp = tp(i, k))))))$$

and  $CNST1.i$  is defined as

$$i.ghost \Rightarrow JC.i \vee (\exists k : S'.i.k)$$

To prove Claim 1, we first prove as follows that, starting at an arbitrary state  $q_k$ , the system reaches a state where  $(\forall i : CNST0.i)$  holds:

- Starting at an arbitrary state  $q_k$ , the system reaches a state where  $(\forall i : i.tp \Rightarrow i.ghost)$  holds within constant time.

For a node  $i$  where  $i.tp \neq \emptyset \wedge \neg i.ghost$  holds, action  $UW$  is enabled if  $R'.i = false$  (since  $CC.i$  holds) and action  $SW$  is enabled if  $R'.i = true$ . Therefore,  $i$  executes  $UW$  within  $d_u$  time or executes  $SW$  within  $d_s$  time, as a result of which  $i.tp$  is set to  $\emptyset$  and thus  $i.tp \Rightarrow i.ghost$  becomes hold for  $i$ .

- Starting at an arbitrary state where  $(\forall i : i.tp \Rightarrow i.ghost)$  holds, the system reaches a state where  $(\forall i : CNST0.i)$  holds within constant time.

For a node  $i$  where  $i.tp \neq \emptyset \wedge \neg TP.i \wedge \neg(\exists k : (S.i.k \wedge i.tp = aspath(i, k)) \vee (S'.i.k \wedge i.tp = tp(i, k)))$  holds, action  $UW$  is enabled at  $i$  since  $CC.i = true$ . Therefore,  $i$  executes  $UW$  within  $d_u$  time, as a result of which  $i.tp = \emptyset$  and thus  $CNST0.i$  holds.

Then we prove that, starting at an arbitrary state where  $(\forall i : CNST0.i)$  holds, the system reaches a state in  $(\forall i : CNST0.i \wedge CNST1.i)$  within constant time:

For a node  $i$  where  $CNST0.i \wedge i.ghost \wedge \neg JC.i \wedge \neg(\exists k : S'.i.k)$  (i.e.,  $CNST0.i = true$  but  $CNST1.i = false$ ), action  $UW$  is enabled since  $LC.i = true$ . Therefore,  $i$  executes  $UW$  within  $d_u$  time, as a result of which  $CNST0.i \wedge CNST1.i$  holds.

Therefore, Claim 1 holds; then we prove that [Claim 2] starting at an arbitrary state  $q'_k$  in  $\mathcal{L}_0$ , the system reaches a state in  $\mathcal{L}$  within  $O(\Gamma(|V'|))$  time, where  $\Gamma(|V'|)$  depends on the route ranking policy used in a network and is the diameter of the timed dispute digraph of the system [?].

To prove Claim 2, we first prove that, starting at an arbitrary state in  $\mathcal{L}_0$  where  $\mathcal{L}$  does not hold, action  $SW$  will be executed at a node within constant time. In a state  $\mathcal{L}_0$  where  $\mathcal{L}$  does not hold, there can be two cases:

- There is no node  $i$  such that  $i.ghost = true$ :  
In this case, there must be a node  $i$  such that  $(\exists k : S.i.k) \vee R.i$  holds (otherwise, the system would be in state  $\mathcal{L}$ ). Therefore, node  $i$  will execute action  $SW$  within  $d_s$  time, unless another neighboring node executes  $SW$  which disables  $(\exists k : S.i.k) \vee R.i$ . Either way, there is at least a node executing  $SW$  within  $d_s$  time.
- There is at least one node  $i$  such that  $i.ghost = true$ :

At a state in  $\mathcal{L}_0$  where there is some node  $i$  such that  $i.ghost = true$ , there must be a node  $j$  such that  $(\exists k : S.j.k) \vee R.j$  holds (otherwise, there would be no node in any containment wave). Therefore, node  $j$  will execute action  $SW$  within  $d_s$  time, unless another neighboring node executes  $SW$  which disables  $(\exists k : S.j.k) \vee R.j$ . Either way, there is at least a node executing  $SW$  within  $d_s$  time.

Thus, starting at a state  $q'_k$  in  $\mathcal{L}_0$ , there will be an action  $SW$  executed within every constant time until the system reaches a state in  $\mathcal{L}$ . By theorems in [?], if an action  $SW$  is executed at time  $t$ , which leads to a new system state  $q_{k''}$ , then there must exist a path from  $q'_k$  to  $q_{k''}$  in the Timed Dispute Graph  $TDD(G')$  of the system. Therefore, starting at a state  $q'_k$  in  $\mathcal{L}_0$ , the system reaches a state in  $\mathcal{L}$  within  $O(\Gamma(|V'|))$  time, where  $\Gamma(|V'|)$  is the diameter of the timed dispute digraph of the system.

Therefore, Claim 2 holds; to finish the proof, we only need to prove that

[Claim 3] once a system reaches a state in  $\mathcal{L}$ , the system remains in  $\mathcal{L}$  in the absence of faults.

Claim 3 trivially holds, since none of actions  $SW$ ,  $CW$ , and  $UW$  is enabled at any node at a state in  $\mathcal{L}$ .

Therefore, Lemma 1 holds for cases where policies other than the SPF policy are used. That is, starting at an arbitrary state, every computation of a system where CPV is used is guaranteed to reach a state in  $\mathcal{L}$  within  $O(\Gamma(|V'|))$  time, where  $\Gamma(|V'|)$  is the diameter of the Timed Dispute Digraph  $TDD(G')$  of the system (which depends on the route ranking policy used).

□

For cases where faults keep occurring at high frequencies, we have

**Theorem 1 (Continuous containment)** *In a system where CPV is used, the contamination range  $R(S_p, q_k)$  of every perturbed region  $S_p$  at an arbitrary state  $q_k$  is  $O(|S_p|)$ . That is, a system where CPV is used is  $\mathcal{F}$ -containing, with  $\mathcal{F}$  being a linear function.*

**Proof:** To prove this theorem, we first consider the case where there is only one perturbed region in the system, then we consider the case where multiple perturbed regions co-exist.

[Single perturbed region] In CPV, for every node  $i$ , only action  $SW$  and  $CW$  could propagate the effect of faults, since only stabilization and containment waves propagate the state changes at a node away from the node (whereas undo-containment waves only undo mistakenly-propagated containment waves, and undo-containments waves do not maintain any variable at all). Therefore, to prove this theorem when there is a single perturbed region, we need to prove that, for any perturbed region  $S_p$ , its contamination range is always  $O(\Gamma(|S_p|))$  irrespective the execution and the system history prefix. There are two cases that can propagate the impact of the perturbed region during protocol execution that tries to stabilize the network from the perturbed state by the faults; we prove that in neither case will the propagation be unbounded.

- Case one: there is a node  $i$  that has a neighbor in  $S_p$ , and  $i$  has just initiated a containment wave (e.g., due to fast state oscillation) when  $i$  turns out not to initiate any stabilization wave.

For every such node  $i$ , action  $UW$  is enabled since  $LC.i = true$ . Therefore, an undo-containment wave will be initiated that propagates along the same direction/paths as the containment wave, as shown in Figure ??.

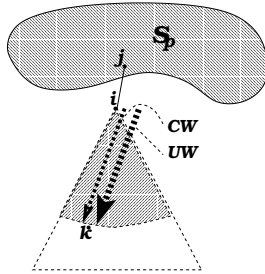


Figure 4: Containment wave in the absence of accompanying stabilization wave

Because the propagation speeds of different diffusing waves are controlled by introducing delays in action execution when a node schedules its enabled actions, and there is difference between clock rates at different nodes,  $d_c > \alpha \cdot (d_u + U)$  should hold in or-

der for the undo-containment wave to catch up with the containment wave. Under this condition, the farthest distance  $dt$  (in terms of the number of hops) the containment wave could propagate satisfies the following formula:  $(U + d_u) \times dt + C0 = (L + d_c) \times dt$  ( $C0$  is the delay in  $i$  initiating the undo-containment wave), thus  $dt = \frac{C0}{d_c - d_u + L - U}$  and is a constant.

Therefore, the theorem holds in this case.

- Case two: there is a node  $i$  that has a neighbor in  $S_p$ , and  $i$  has initiated a containment wave  $CW0$  because  $i$  needs to initiate a stabilization wave  $SW0$  by protocol CPV (see Figure ??).

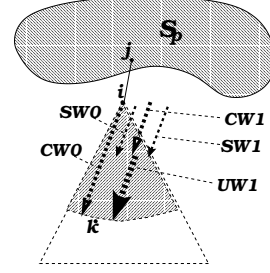


Figure 5: Consistent containment and stabilize waves

In the presence of continuously occurring faults, there are two subcases:

- If  $i$  needs to change state after another fault  $f$  occurs, then every node whose route goes through  $i$  will need to change state anyway, no matter whether there is another stabilization wave  $SW0'$  other than  $SW0$  to be propagated by  $i$ .

In this case, all the nodes that  $CW0$  could have reached are perturbed nodes and need to change state. Therefore, the existence of  $CW0$  and  $SW0$  does not introduce any contaminated nodes, and the theorem trivially holds.

- If  $i$  does not need to change state in the presence of faults that may occur to nodes in  $S_p$  later, then action  $CW$  and  $SW$  will be enabled at  $i$  later when nodes in the perturbed region change their state. The latest for this to occur is at most  $O(|S_p|)$  time after  $SW0$  is initiated by  $i$ , since a containment wave must be initiated in  $S_p$  and it takes at most  $O(|S_p|)$  time for the containment wave to reach  $i$  (note that the hop-length of the longest path segment in  $S_p$  is  $|S_p|$ ).

When action  $CW$  is enabled  $i$ , a containment wave  $CW1$  will be initiated/propagated which propagates along the same paths taken by  $CW0$  and  $SW0$ . Given that containment waves propagate faster than stabilization waves do,  $CW1$  will catch up with and stop  $SW0$ , at which point an undo-containment wave  $UW1$

will be initiated to catch up with  $CW0$ . When node  $i$  executes action  $SW$ ,  $i$  initiates another stabilization wave  $SW1$  that stabilizes the state of nodes which have been affected by  $SW0$  and  $CW1$ . No matter when  $SW1$  will be initiated, the maximum distance  $CW0$  and  $SW0$  could propagate is bounded by the point where  $UW1$  catches up with  $CW0$ .

In order for  $CW1$  to be able to catch up with and stop  $SW0$  in the above process,  $d_s > \alpha \cdot (d_c + U)$  should hold, given that the propagation speeds of different diffusing waves are controlled by introducing delays in action execution when a node schedules its enabled actions, and that there is difference between clock rates at different nodes. Therefore, the farthest distance  $dt$  that  $CW0$  and  $SW0$  could propagate before being stopped satisfies the following formula:  $O(|S_p|) + (U + d_c) \times dt \times \frac{U+d_c}{L+d_s} + (U + d_u) \times dt \times (1 - \frac{U+d_c}{L+d_s}) = (L + d_c) \times dt$ , thus  $dt = O(|S_p|)$ .

Therefore, the theorem holds in this case.

**[Multiple perturbed regions]** When there are multiple perturbed regions, we need to prove that there is no interference between different perturbed regions. Toward this end, we first define the contamination region of a perturbed region  $S$  as the set of nodes that are contaminated by  $S$ .

Then, if the the contamination regions of two perturbed regions are disjoint, the claim holds trivially since they are unable to affect each other at all. Therefore, we only consider the case where the contamination regions of two or more perturbed regions are adjoining one another.

Without loss of generality, let us consider two perturbed regions  $S0$  and  $S1$ , whose contamination regions are  $CS0$  and  $CS1$  respectively, and  $CS0$  and  $CS1$  are adjoining each other. To prove that  $S0$  and  $S1$  do not interfere each other, we only need to prove that, without loss of generality, the containment patterns as shown in Figures ?? and ?? for  $S0$  are not affected by any node  $k0$  that is contaminated by  $S0$  but has a neighbor  $k1$  that is contaminated by  $S1$ . This is the case because

- On one hand, if  $k0$  does not change state because of the state changes at  $k1$ , then the existence of  $S1$  does not affect the protocol execution within the contamination region of  $S0$ , and thus  $S1$  does not affect the containment activities (by the execution of action  $CW$  and  $UW$  in the contamination region of  $S0$ ) for  $S0$ ;
- On the other hand, if  $k0$  changes state because of the state changes at  $k1$  and this interrupts the execution of containing action  $CW$  or  $UW$  in the contamination region of  $S0$ , this must be due to the fact that the corresponding protocol action  $CW$  or  $UW$  becomes disabled at  $k0$  due to the state changes at

$k1$ . Therefore,  $k0$  is regarded as contaminated by  $S1$ , and  $k0$  will be involved in the containing action  $CW$  or  $UW$  in the contamination region of  $S1$ . Thus, the state changes at  $k0$  will be contained too even in this case.

Therefore, the theorem holds when there are multiple perturbed regions at a state.  $\square$

For cases where a node keeps changing its state (e.g., keeps fail-stopping and rejoining [5, 12, 19]), we have

**Theorem 2 (Stability-adaptive control)** *In a system where CPV is used, the distance to which a state  $q_{i,k}$  of a node  $i$  propagates is  $\theta(t_{i,k})$ , where  $t_{i,k}$  is the sojourn time of state  $q_{i,k}$ .*

**Proof:** To prove this theorem, let us consider an arbitrary pair of consecutive states  $q_{i,k}$  and  $q_{i,k+1}$  for a node  $i$  where the sojourn time for  $q_{i,k}$  is  $t_{i,k}$ .

When  $i$  changes to state  $q_{i,k}$ ,  $i$  initiates a stabilization wave  $SW_{i,k}$  (in addition to a containment wave  $CW_{i,k-1}$  that is to contain the propagation of the immediately previous state to  $q_{i,k}$  at node  $i$ ). When  $i$  changes from state  $q_{i,k}$  to  $q_{i,k+1}$  after  $t_{i,k}$  time,  $SW_{i,k}$  would have propagated up to  $\frac{t_{i,k}}{d_s+L}$  hops. When  $i$  changes to state  $q_{i,k+1}$ ,  $i$  initiates a containment wave  $CW_{i,k}$  to catch up with  $SW_{i,k}$ . The farthest distance  $dt$  from a node where  $CW_{i,k}$  catches up with  $SW_{i,k}$  to  $i$  satisfies formula:  $dt \times (d_s + L) = t_{i,k} + dt \times (d_c + U)$ . Therefore,  $dt = \frac{t_{i,k}}{d_s - d_c + L - U} = \theta(t_{i,k})$ . Similarly, the shortest distance  $dt'$  from a node where  $CW_{i,k}$  catches up with  $SW_{i,k}$  to  $i$  satisfies formula:  $dt' \times (d_s + U) = t_{i,k} + dt' \times (d_c + L)$ . Therefore,  $dt = \frac{t_{i,k}}{d_s - d_c + U - L} = \theta(t_{i,k})$ .

Thus, the theorem holds.  $\square$

By Theorem 2, we see that, in CPV, the more unstable a node is, the shorter is the distance to which its state propagates (since the sojourn time of the state of a more unstable node is shorter). And network stability is improved without sacrificing network convergence in the sense that *a steady state reaches outwards, and a transient state stays*.

For cases where faults stop occurring (either indefinitely or for a long enough period) after a point in time, we have

**Theorem 3 (Local stabilization)** *Starting at an arbitrary state  $q_k$  with an arbitrary history prefix  $\mathcal{H}(q_k)$  and an arbitrary protocol execution  $\mathcal{E}(q_k)$ , the system computation where CPV is used reaches a legitimate state within  $O(\Gamma(\mathcal{P}(q_k, \mathcal{H}(q_k), \mathcal{E}(q_k))))$  time in the absence of faults, where  $\Gamma$  is a function reflecting the route ranking policy used in the system. That is, a system where CPV is used is  $\Gamma$ -stabilizing.*

**Proof:** Similar to the proof for Theorem 1, we prove this theorem by considering the two different cases depending on whether or not there is only one perturbed region in the system.

[Single perturbed region] When there is only one perturbed region  $S_p$  in the network, the time taken for nodes in  $S_p$  to stabilize is  $O(\Gamma(|S_p|))$  according to the proof for Lemma 1 and Theorem 1.

By Theorem 1, the contamination range of  $S_p$  is  $O(\Gamma(|S_p|))$ . Since the time taken for the nodes in the contamination region of  $S_p$  to stabilize is proportional to the contamination range of  $S_p$ , the time taken for nodes contaminated by  $S_p$  to stabilize is  $O(\Gamma(|S_p|))$ .

When the time taken for the system to stabilize is  $O(\Gamma(|S_p|)) + O(\Gamma(|S_p|)) = O(\Gamma(|S_p|))$ . When  $S_p$  is the only perturbed region in the network, the perturbed size  $\mathcal{P}(q_k, \mathcal{H}(q_k), \mathcal{E}(q_k)) = |S_p|$ . Therefore, the system stabilizes within  $O(\Gamma(\mathcal{P}(q_k, \mathcal{H}(q_k), \mathcal{E}(q_k))))$  time, and the claim holds in this case.

[Multiple perturbed regions] When there are multiple perturbed regions  $S.0, S.1, \dots$ , and  $S.m$ , the time taken for the network to stabilize depends on whether the contamination regions of these perturbed regions are disjoint or not.

When the contamination regions of  $S.0, S.1, \dots$ , and  $S.m$  are disjoint, the stabilization of each contamination region is independent of that of the other contamination regions. Therefore, the time taken for the network to stabilize is  $O(\max_{i=0..m} \Gamma(|S.i|))$ . Since  $\mathcal{P}(q_k, \mathcal{H}(q_k), \mathcal{E}(q_k)) = \sum_{i=0}^m |S.i|$  and  $\Gamma(\sum_{i=0}^m |S.i|) \geq \max_{i=0..m} \Gamma(|S.i|)$  (by the definition of Timed Dispute Graphs [?]), the network stabilizes within  $O(\Gamma(\mathcal{P}(q_k, \mathcal{H}(q_k), \mathcal{E}(q_k))))$  time.

When the contamination regions of two perturbed regions  $S.k$  and  $S.(k+1)$  are adjoining, the stabilization of one perturbed-region may depend on that of the other, because the stabilization in path-vector routing is essentially a diffusing computation from the destination. That is, one perturbed region as well as its contamination region can stabilize only after the other perturbed region as well as its contamination region has stabilized. Therefore, in the worst case where the set of perturbed regions as well as their contamination regions stabilize sequentially, the time taken for the system to stabilize is the summation of the stabilization time for each perturbed regions, i.e.,  $O(\sum_{i=0}^m \Gamma(|S.i|))$ . Given that  $\mathcal{P}(q_k, \mathcal{H}(q_k), \mathcal{E}(q_k)) = \sum_{i=0}^m |S.i|$  and  $\Gamma(\sum_{i=0}^m |S.i|) \geq \sum_{i=0}^m \Gamma(|S.i|)$  (by the definition of Timed Dispute Graphs and the fact that  $\Gamma$  represents the length of the longest simple path in the Timed Dispute Graph [?]), the network stabilizes within  $O(\Gamma(\mathcal{P}(q_k, \mathcal{H}(q_k), \mathcal{E}(q_k))))$  time.

Therefore, the theorem holds.  $\square$

In the common case where the SPF policy is used, Theorem 3 implies

**Corollary 1** *A system where CPV and the SPF policy are used is  $\Gamma$ -stabilizing, and  $\Gamma$  is a linear function.*

**Proof:** From the proof for Theorem 3, we see that the time taken for a perturbed region  $S_p$  as well as its con-

tamination region to stabilize is determined by the time taken for  $S_p$  to stabilize, which is  $O(\Gamma(|S_p|))$  in general and  $\Gamma$  is a function depending on the route ranking policy used in the system. By the proof for Lemma 1, we see that, when the SPF policy is used, the time taken for a perturbed region  $S_p$  to stabilize is  $O(|S_p|)$ . Therefore, this corollary trivially holds.  $\square$

By the analysis above, we see that CPV contains continuously occurring faults and locally stabilizes; the degree of fault containment and the time taken to stabilize is a function of the perturbation size instead of the network size. We also see that CPV is “stability-adaptive” in the sense that the state of stable nodes propagate outwards, and the state of unstable nodes is locally contained.

In the next section, we corroborate our analysis by simulating Internet-like networks and studying how continuous containment and local stabilization improve packet-forwarding in the presence of continuously occurring faults.

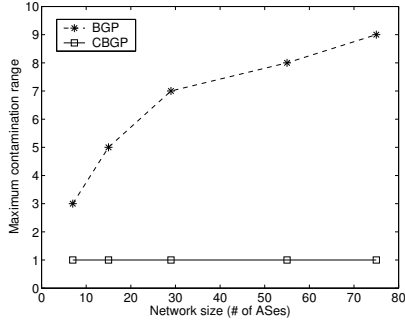
## 6 Simulation results

We have implemented CPV in SSFNet [1], a network simulator which supports a rich set of standard Internet protocols such as BGP (with route-flap-damping). In Section 5, we have analyzed continuous containment and local stabilization in CPV; in this section, therefore, we focus on the impact of continuous containment and local stabilization on packet forwarding in the presence of continuously occurring faults.

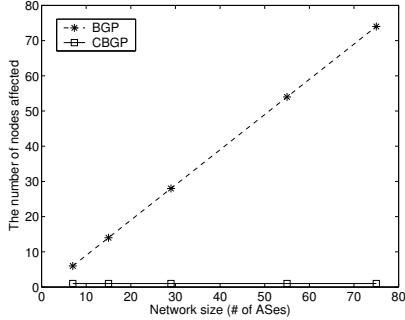
In our simulation, CPV is executed without using any existing instability-suppression mechanisms; BGP is executed with instability-suppression timers and route-flap-damping with the standard parameter setup. For comparability, we set parameter  $d_s$  of CPV as 30 seconds, which is the default MinRouteAdvertisementInterval value used in BGP. Accordingly, we set  $d_c$  and  $d_u$  as 10 and 1 seconds respectively for CPV.

For fidelity of simulation, we use realistic Internet-type topologies [1] to evaluate CPV. And to study the impact of network size, we use networks of size ranging from 7 ASes to 75 ASes. To simulate continuously occurring faults, we let an arbitrary node  $j$  repeatedly fail-stop and then rejoin every 30 seconds. The simulation results are as follows.

**Continuous containment.** Figure 4 shows the maximum contamination range and the maximum number of nodes affected (i.e., perturbed or contaminated) by the faults in BGP and CPV. We see that the contamination range and the number of nodes affected by the faults in BGP increase as network size increases, and every node is affected by the faults in BGP; whereas in CPV, the contamination range remains 1 and the number of nodes affected by the faults remains small as network size in-



(a) Maximum contamination range



(b) The number of nodes affected

Figure 6: Impact of continuously occurring faults

creases, since CPV contains the impact of the continuously occurring faults locally around where they occur (as also proved in Theorem 1).

**Local stabilization.** Figure 5 shows the time taken for BGP and CPV to stabilize once faults stop occurring when  $j$  is up (we discuss the case where a node fail-stops in Section 7). We see that the time taken for BGP to

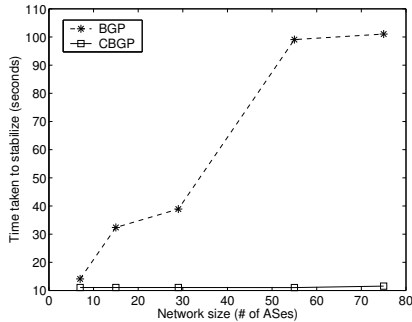


Figure 7: Time taken to stabilize

stabilize increases as network size increases; whereas the time taken for CPV to stabilize remains small as network size increases, since the time taken for CPV to stabilize depends on the perturbation size instead of the network

size (as also proved in Theorem 3), and the perturbation size remains small in CPV as network size increases.

**Stability-adaptive control.** To study the property of stability-adaptiveness in CPV, we let an arbitrary node oscillate among different states and study the relationship between the sojourn time of a state and the distance to which the state propagates (i.e., the maximum distance from the oscillating node to the nodes that adapt to the state). The results are shown in Figure 8. We

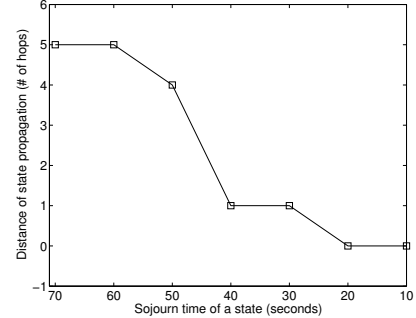


Figure 8: Stability-adaptiveness of CPV

see that states that last longer propagate farther and, accordingly, more nodes adapt behaviors; as the sojourn time of a state becomes shorter and shorter, the distance to which the state propagates decreases more and more (as also proved in Theorem 2).

## 7 Discussion

In this section, we discuss approaches to further improve the performance of CPV, mechanisms to implement CPV efficiently, and incremental deployment of CPV.

**Sub-linear containment & stabilization.** We mainly focused on the containment of continuously occurring faults in this paper, and the contamination range as well as the convergence time (after faults stop occurring) is a linear function of the perturbation size in the common case where the SPF policy is used. If we apply, together with CPV, mechanisms that expedite the convergence of path-vector routing protocols (such as those proposed in [13] and [20]), then the contamination range of a perturbed-region is reduced to a linear function of the diameter of the region, and thus both the contamination range and the convergence time are reduced to a sub-linear function of the perturbation size.

**Implementation of CPV.** In CPV, neighboring nodes exchange their state periodically, which is required for protocols to stabilize from state corruptions. To reduce the overhead of the periodic information synchronization, we can apply the technique proposed in [18] that guarantees the consistency of the routing-tables between neighboring nodes in a scalable manner.

In CPV, when network state changes, a node  $i$  maintains variable  $i.tp$  to denote the next route that  $i$  will

adopt. Since  $i.tp$  is transient and is meaningful only during the stabilization of CPV, each node  $i$  only needs to maintain  $i.tp$  for destinations to which the route of  $i$  needs to change. Therefore, the use of  $i.tp$  in CPV does not introduce much memory overhead, since the majority of the network is expected to be stable most of the time.

**Incremental deployment of CPV.** Containment waves in CPV introduce new information other than that used in BGP. To enable graceful migration of and interoperability with BGP, we define a new *optional transitive* path attribute [16], and use this attribute to encode the information newly introduced in CPV. According to BGP specification, a BGP speaker propagates a transitive attribute upon receipt, whether or not the BGP speaker implements CPV. Therefore, CPV can be incrementally deployed and inter-operate well with BGP. Moreover, even in the case of partial deployment, CPV can help contain faults that keep occurring to a region where CPV is used.

## 8 Concluding remarks

To characterize system properties in the presence of continuously occurring faults, we formulated the notions of perturbed node, contaminated node, perturbation size, contamination range,  $\mathcal{F}$ -containment, and  $\mathcal{F}$ -stabilization. These concepts are generically applicable to networking and distributed computing problems. In general, a self-stabilizing protocol that contains continuously occurring faults also locally stabilizes, and a locally stabilizing protocol contains faults that occur only once and at the same time.

We designed the path-vector routing protocol CPV that contains continuously occurring faults and locally stabilizes. In CPV, the distance to which the state of a node propagates is proportional to the sojourn time of the state. CPV achieves these properties by layering a system computation into three diffusing waves (i.e., stabilization wave, containment wave, and undo-containment wave) which run in parallel and coordinate to contain the propagation of obsolete information while stabilizing a network at the same time. Built upon models for the Internet, CPV is readily applicable.

In this paper, we focused on how to contain high-frequency unanticipated faults without detecting why and where the faults occur. This design enables CPV to work both when faults are benign and transient instability is unavoidable [20] as well as in adversarial cases where nodes are compromised [12, 19]. In the latter cases, CPV can be applied together with security and fault diagnostic mechanisms that detect and suppress mis-behaving nodes or links; since CPV contains faults locally around where they occur, the security and fault diagnostic mechanisms can be more optimistic (e.g., higher cutoff threshold and shorter suppression time in

route flap damping [17]) to avoid issues such as severely delayed convergence in BGP after route flap damping.

## Acknowledgment

We thank Zhijun Liu for his help in implementing a preliminary version of CPV in SSFNet. We also thank Dan Massey for discussions on issues with BGP.

## References

- [1] Modeling the global internet. In <http://www.ssfnet.org/>.
- [2] A. Arora and H. Zhang. LSRP: Local stabilization in shortest path routing. In *IEEE-IFIP DSN*, pages 139–148, June 2003.
- [3] Y. Azar, S. Kutten, and B. Patt-Shamir. Distributed error confinement. In *ACM PODC*, pages 33–42, 2003.
- [4] A. Bremner-Barr, Y. Afek, and S. Schwarz. Improved BGP convergence via ghost flushing. In *IEEE INFOCOM*, 2003.
- [5] D.-F. Chang, R. Govindan, and J. Heidemann. An empirical study of router response to large BGP routing table load. In *ACM SIGCOMM-USENIX IMW*, pages 203–208, 2002.
- [6] J. Cowie, A. T. Ogielski, B. Premore, and Y. Yuan. Internet worms and global routing instabilities. In *SPIE*, 2002.
- [7] S. Ghosh, A. Gupta, T. Herman, and S. V. Pemmaraju. Fault-containing self-stabilizing algorithms. In *ACM PODC*, pages 45–54, 1996.
- [8] M. G. Gouda. *Elements of Network Protocol Design*. John Wiley and Sons, 1998.
- [9] C. Labovitz, G. R. Malan, and F. Jahanian. Origins of internet routing instability. In *IEEE INFOCOM*, pages 218–226, 1999.
- [10] J. Luo, J. Xie, R. Hao, and X. Li. An approach to accelerate convergence for path vector protocol. In *IEEE GLOBECOM*, pages 2390 – 2394, 2002.
- [11] R. Mahajan, D. Wetherall, and T. Anderson. Understanding BGP misconfiguration. In *ACM SIGCOMM*, pages 3–16, 2002.
- [12] NISCC. Vulnerability issues in TCP. In *NISCC Vulnerability Advisory 236929*, April 2004.
- [13] D. Pei, M. Azuma, N. Nguyen, J. Chen, D. Massey, and L. Zhang. BGP-RCN: Improving BGP convergence through root cause notification. In *Technical Report TR030047, UCLA Computer Science Department*, 2003.
- [14] D. Pei, X. Zhao, D. Massey, and L. Zhang. A study of BGP path vector route looping behavior. In *IEEE ICDCS*, pages 720–729, March 2004.
- [15] D. Pei, X. Zhao, L. Wang, D. Massey, A. Mankin, S. F. Wu, and L. Zhang. Improving BGP convergence through consistency assertions. In *IEEE INFOCOM*, pages 976–985, 2002.
- [16] Y. Rekhter and T. Li. A border gateway protocol 4 (BGP-4). In *IETF RFC 1771*, March 1995.
- [17] C. Villamizar, R. Chandra, and R. Govindan. BGP route flap damping. In *IETF RFC 2439*, November 1998.
- [18] L. Wang, D. Massey, K. Patel, and L. Zhang. FRTR: A scalable mechanism to restore routing table consistency. In *IEEE-IFIP DSN*, 2004, to appear.
- [19] L. Wang, X. Zhao, D. Pei, R. Bush, D. Massey, A. Mankin, S. F. Wu, and L. Zhang. Observation and analysis of BGP behavior under stress. In *ACM SIGCOMM-USENIX IMW*, pages 183–195, 2002.
- [20] H. Zhang, A. Arora, and Z. Liu. A stability-oriented approach to improving BGP convergence. In *IEEE SRDS*, 2004.