Finite State Machines for BGP

Dan Pei[1], Dan Massey[2] and Lixia Zhang[1]

UCLA[1] Colorado State University[2]
{peidan,lixia}@cs.ucla.edu massey@cs.colostate.edu

UCLA CSD Technical Report TR040047

Abstract

This report presents the finite state machines for the BGP decision process, and uses these finite state machines to BGP protocol syntax checking. The routing protocol syntax defines the legitimate sequence of messages and is used to reject invalid messages. If the protocol syntax is well defined, syntax checking can be very effective at detecting hardware faults, implementation bugs, and so forth. To better capture the protocol syntax, from the finite state machines we derive protocol assertions the rules that must hold true when the BGP protocol is functioning properly. This approach formalizes the protocol specification and only legitimate sequence of routing update messages are allowed by the corresponding state machine. Illegitimate message sequences resulting from implementation bugs, hardware faults can be detected by violations of the finite state machine transitions. We demonstrate that some pathological updates(false withdrawals) in the early Internet [2] would've been detected should one of our assertion checking had been deployed.

I. Introduction

The routing protocol syntax defines the legitimate sequence of messages and is used to reject invalid messages. A common approach taken by routing protocols is to use heart beat messages to detect whether a neighbor is reachable (i.e. Hello in OSPF, Keep-Alive in BGP). BGP neighbor to neighbor peering also uses extensive syntax checking. However, broader syntax checking involving more than peer to peer communication is seldom used in a distributed system such as a routing protocol.

If the protocol syntax is well defined, syntax checking can be very effective at detecting hardware faults, implementation bugs, and so forth. We use the protocol specification to construct finite state machines for BGP. This approach formalizes the protocol specification and only legitimate sequence of routing update messages are allowed by the corresponding state machine. Illegitimate message sequences resulting from implementation bugs, hardware faults can be detected by violations of the finite state machine transitions.

For example, the BGP protocol uses a withdrawal update message to notify neighbors when a prefix is no longer reachable. Intuitively, a router should not withdraw a route it has never advertised. However, [2] measured BGP updates and observed that a majority of BGP updates in the Internet were pathological update such as withdrawal messages for routes that were never advertised. Furthermore, a popular router implementation would further propagate the withdrawal messages. One of our assertions(Assertion 3) derived from our Finite State Machine would immediately identify this type of illegitimate syntax and would have detected these pathological updates.

II. State Machines for one single prefix

In this section, we develop Finite State Machines for BGP protocol[3]. A state machine for the whole decision process of a BGP speaker is quite complex, and we would like to take as our first step developing state machines for a BGP speaker's processing of a single prefix, without considering aggregation. Furthermore, we divide the state machines into three parts of communicating state machines[1], illustrated in Figure 1.

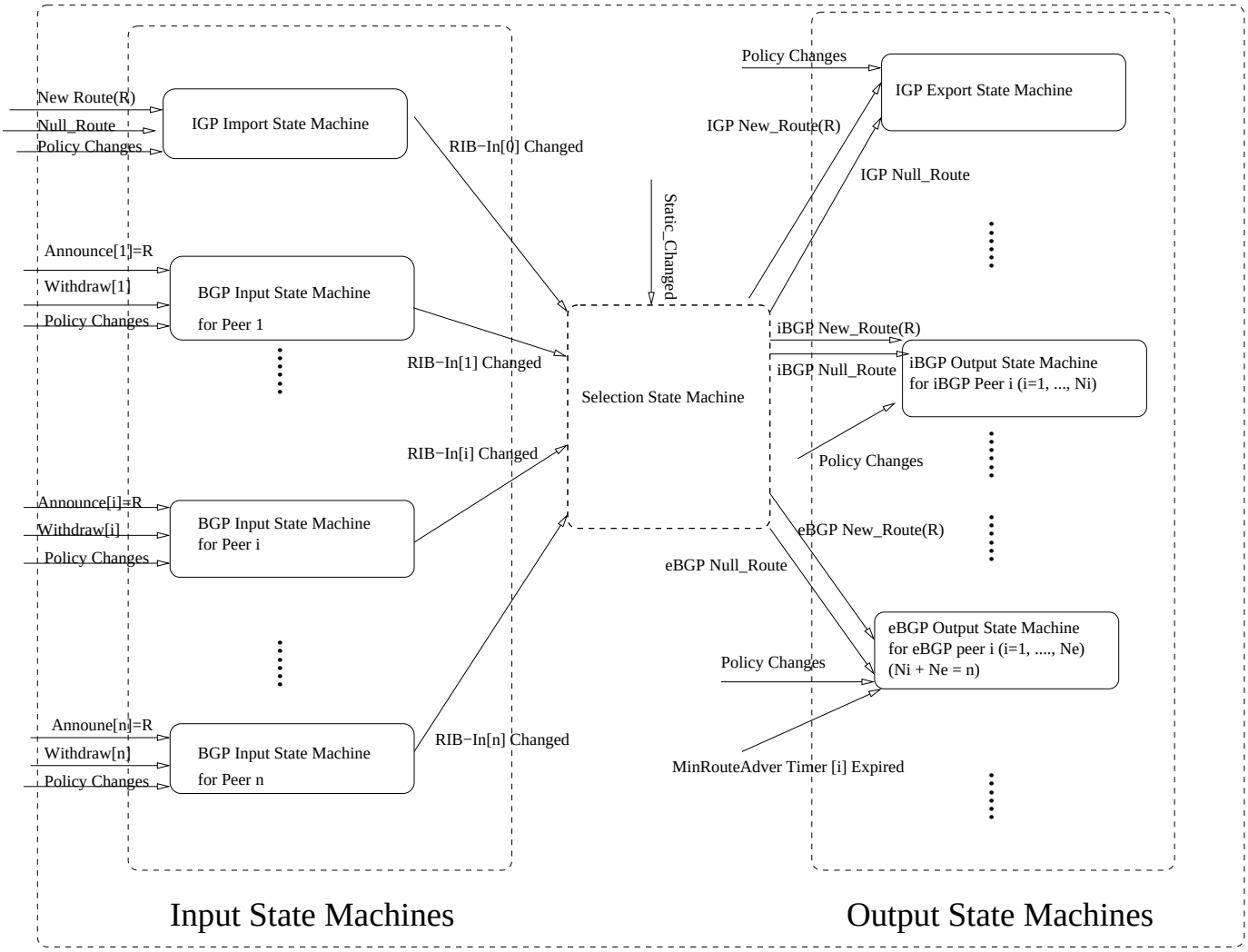


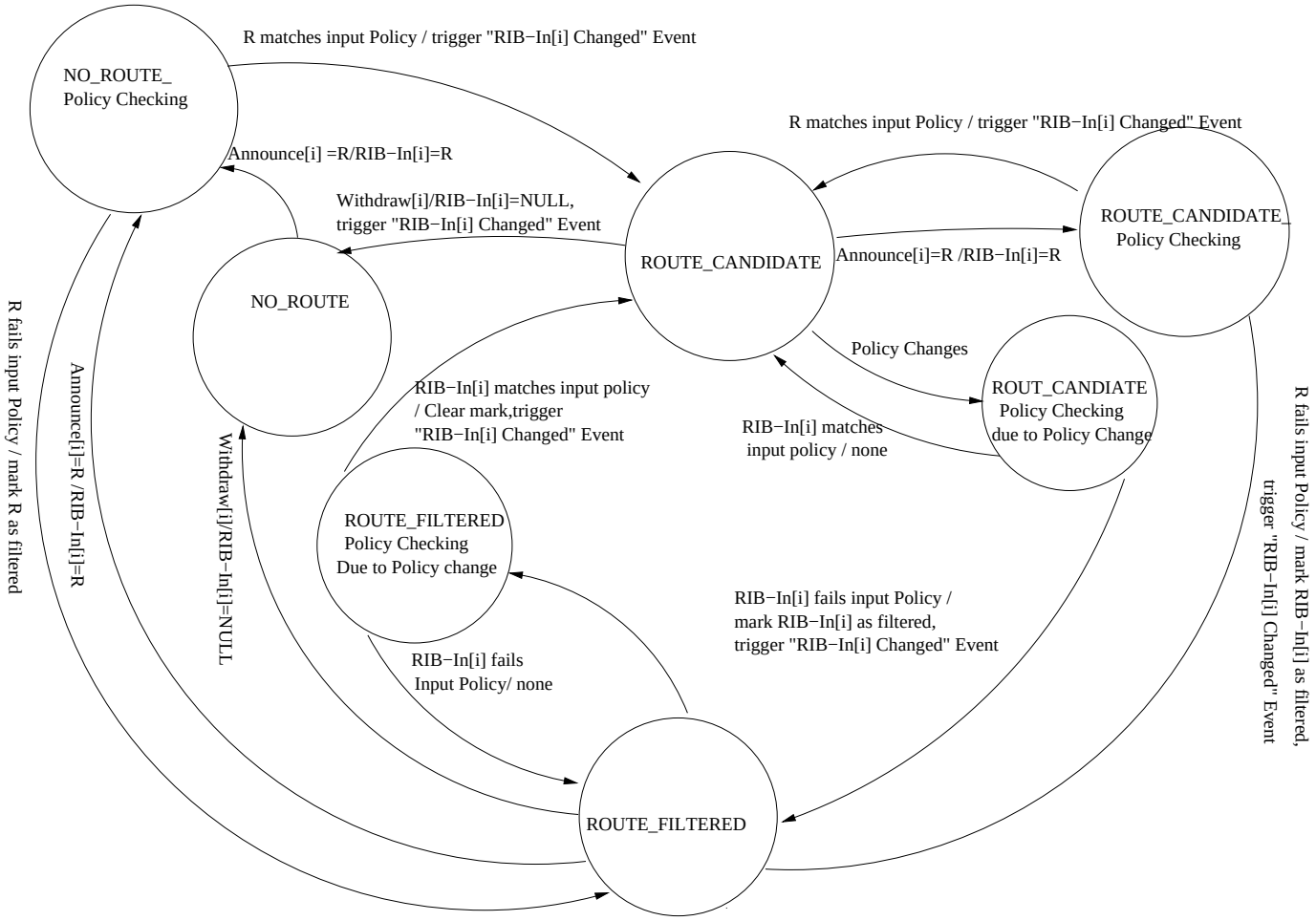
Fig. 1. Relationship of the State Machines

These state machines include input State Machines, a Selection State Machine(Figure 4, and Output State Machines. Input State Machines include one IGP Import State Machine(Figure 3), and one BGP Input State Machine for each peer(Figure 2). Output State Machines include one iBGP Output State Machine for each iBGP peer(Figure 5), one eBGP Output State Machine for each eBGP peer(Figure 6) and Timer State Machine for each eBGP peer(Figure 7). These state machines interact with each other through the events they trigger.

III. Assertions Derived from Finite State Machines

Following the transition rules of state machines, the outputs of these state machines are correct if the inputs are correct. Therefore, an incorrect output is a clear indication that something is wrong in the input, assuming the state machines are implemented correctly. Our approach is based on this observation, and derives *protocol assertions* that must hold true in any correct execution of the protocol. Figure 8 lists the assertions that derived from state machines.

We classify the assertions into following 4 types:



Input Events:

"Announce[i]=R" Event: triggered by the receiving of an announcement for prefix p from peer i

"Withdraw[i]" Event: triggered by the receiving of a withdrawal for prefix p from peer i, or triggered by the loss of the BGP session with peer i

"Policy Changes" Event: triggered when the Policy is changed in user interface

Output Events:

"RIB−In[i] Changed" Event, which will be used by the Selection State Machine

Internal Events: (Which are triggered by the "Policy Checking" states, and such an event could only be used by the state that trigger it.)

"RIB−In[i] matches Input Policy"

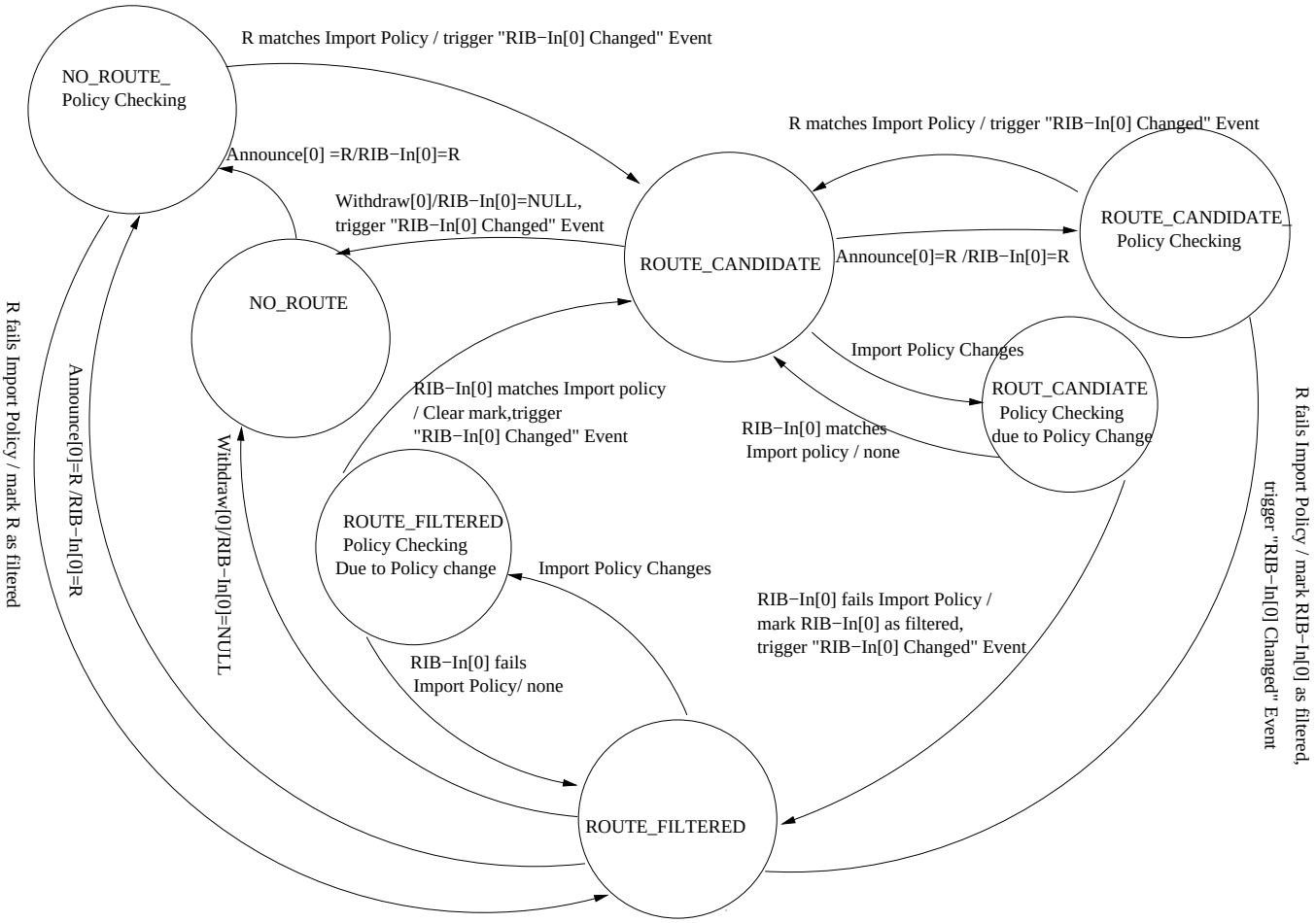
"RIB−In[i] fails Input Policy"

Side Effects: "

Modifications on RIB−In[i],including marking and unmarking a RIB−In[i] as filtered

Note: "Policy Checking" States are intermediate states, and the process that" a trasition goes to state, and a following trasition goes out of it is atomic. Only the Internal Events shown above could occur when the machine is in the "Policy Checking" state.

Fig. 2. Input State Machine for Route to prefix p that is received from BGP peer i.



Input Events:

"New_Route(R)" Event: triggered when a new route R is imported from IGP

"Null_Route" Event: triggered when IGP becomes to have no route to prefix p

"Import Policy Changes" Event: triggered when the Import Policy is changed in user interface.

Output Events:

"RIB−In[0] Changed" Event, which will be used by the Selection State Machine

Internal Events: (Which are triggered by the "Policy Checking" states, and such an event could only be used by the state that triggered it.)

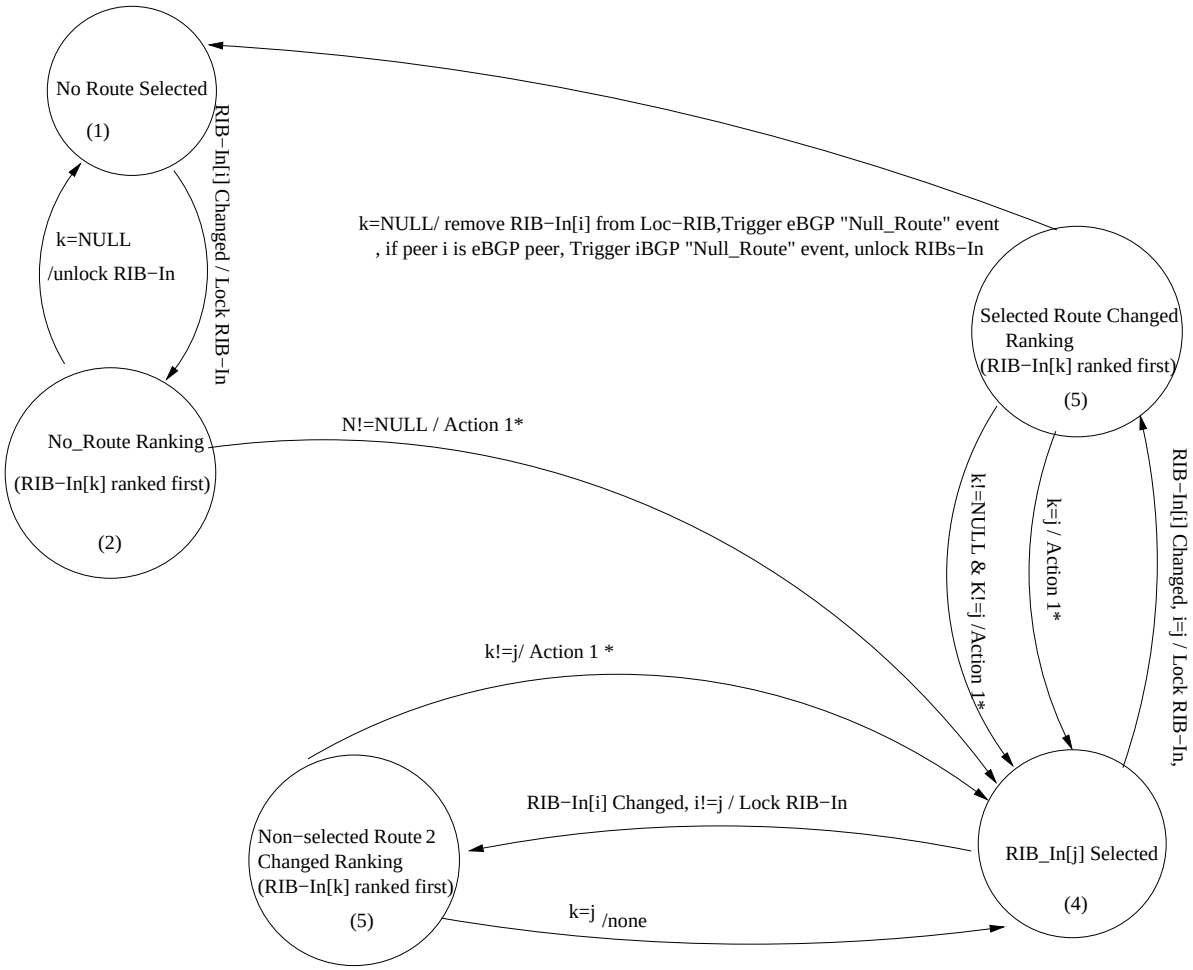
"RIB−In[0] matches Import Policy"

"RIB−In[0] fails Import Policy"

Note: "Policy Checking" States are intermediate states, and the process that" a trasition goes to state, and a following trasition goes out of it " is atomic. Only the Internal Events shown above could occur when the machine is in the "Policy Checking" state.

In order for ease of describtion in Selection State Machine, we use RIB−In[0] to refer to the IGP route to prefix p.

Fig. 3. State Machine for Route to prefix *p* that are imported from IGP



Input Events:
"RIB−In[i]=R" Event: triggered by the Input State Machine for peer i, or triggered by Input State Machine for Static/IGP

Output Events:

eBGP "New_Route(<Route>)" Event: which will be used by eBGP Output State Machine
eBGP "Null_Route" Event: which will be used by eBGP Output State Machine
iBGP "New_Route(<Route>)" Event: which will be used by iBGP Output State Machine
iBGP "Null_Route" Event: which will be used by iBGP Output State Machine

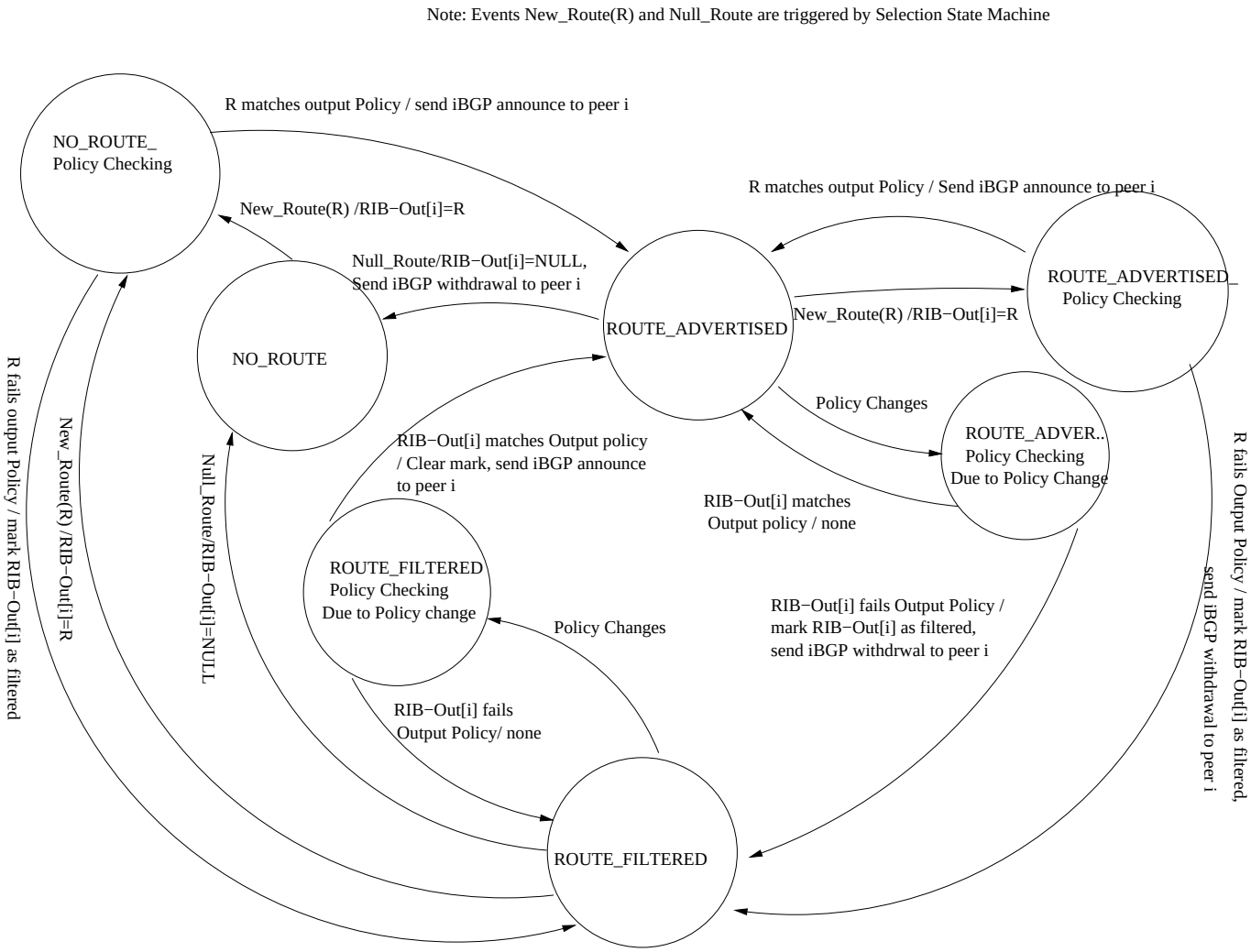
Internal Events: (Which are triggered by the "Ranking" states, and such an event could only be used by the state that triggered it.)
All the other events other than the Input Events and Output Events listed above.

Side Effects:
Modifications on RIB−Loc, FIB, IGP

Note: "Ranking" States are intermediate states, and the process that" a trasition goes to state, and a following trasition goes out of it" is atomic. Only the Internal Events shown above could occur when the machine is in the "Policy Checking" state.

Note* Action 1: install RIB[k] in Loc−RIB, FIB and IGP;
j<−k, unlock RIBs−In; Trigger eBGP "New_Route(RIB−In[j])" event
if peer i is eBGP peer, Trigger iBGP "New_Route(RIB−In[j]" event

Fig. 4. Selection State Machine for Route to prefix p: this state machine describes state changes in a BGP speaker that are associated with the route to prefix p.



Input Events:

"iBGP New_Route(R)" Event: triggered by Selection State Machine
"iBGP Null_Route(R)" Event: triggered by Selection State Machine
"Policy Changes" Event: triggered when the Policy is changed in user interface

Output Events:

None

Internal Events: (Which are triggered by the "Policy Checking" states, and such an event could only be used by the state that trigger it.)

RIB−Out[i] matches Output Policy

RIB−Out[i] fails Output Policy

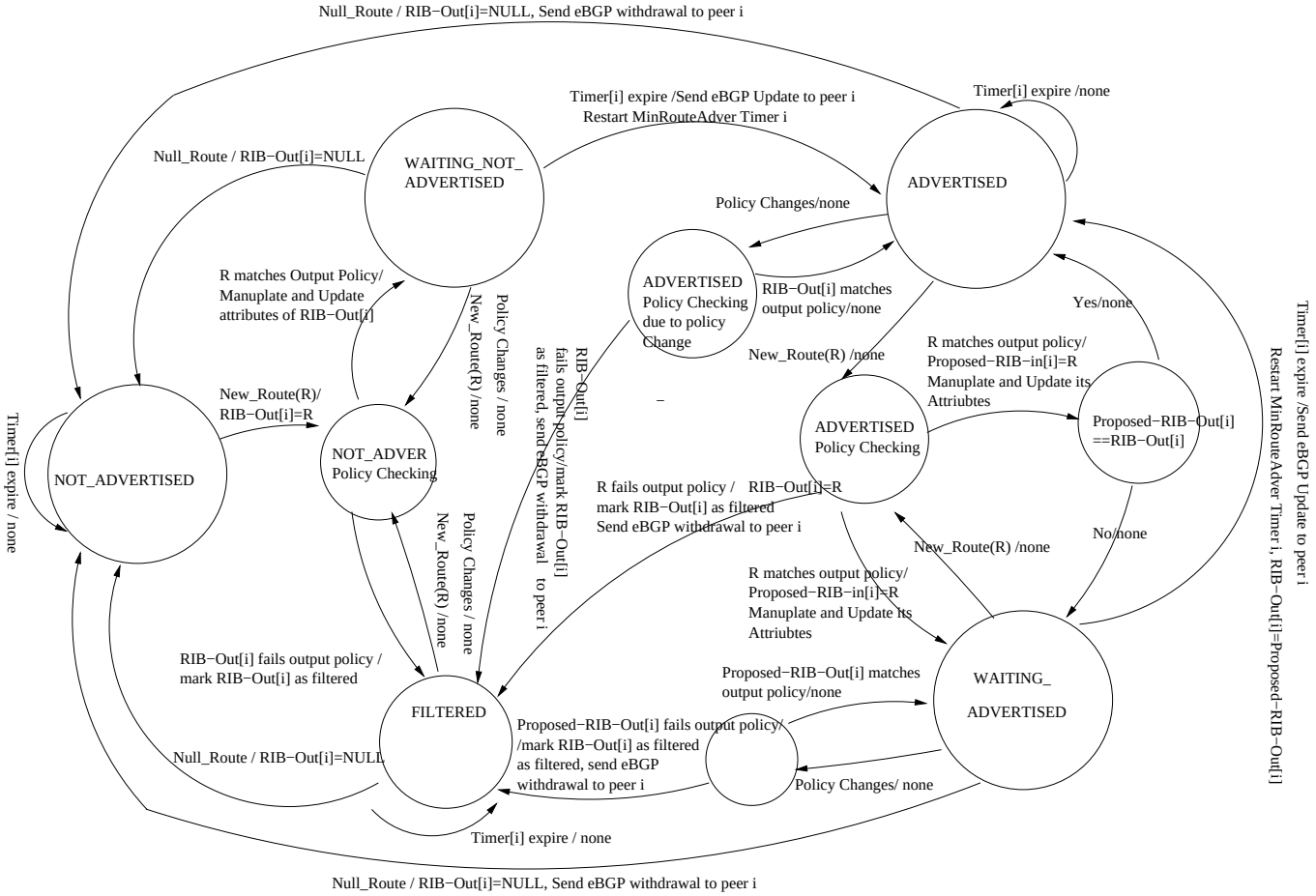
Side effects:

Modification on RIB−Out[i], possibly sending of announcement or withdrawal to peer i

Note: "Policy Checking" States are intermediate states, and the process that" a trasition goes to state, and a following trasition goes out of it" is atomic. Only the Internal Events shown above could occur when the machine is in the "Policy Checking" state.

RIB−Out[i]=R meaning changing the flag.... see Internet Routing Architectures, 2nd. p154.

Fig. 5. Output State Machine for Route to prefix p that will be sent to iBGP peer i



Input Events:

"eBGP New_Route(R)" Event: triggered by Selection State Machine

"eBGP Null_Route(R)" Event: triggered by Selection State Machine

"Policy Changes" Event: triggered when the Policy is changed in user interface

"Timer[i] expired" Event: triggered when the MinRouteAdver timer for peer i expired.

Output Events:

None

Internal Events: (Which are triggered by the "Policy Checking" states, and such an event could only be used by the state that triggered it.)

RIB−Out[i] matches Output Policy

RIB−Out[i] fails Output Policy

Side effects:

Modification on RIB−Out[i], possibly sending of announcement or withdrawal to peer i , restarting MinRouteAdver timer

Note: "Policy Checking" States are intermediate states, and the process that" a trasition goes to state, and a following trasition goes out of it" is atomic. Only the Internal Events shown above could occur when the machine is in the "Policy Checking" state.

RIB−Out[i]=R meaning changing the flag.... see Internet Routing Architectures, 2nd. p154.

Fig. 6. Output State Machine for Route to prefix p that will be sent to eBGP peer i

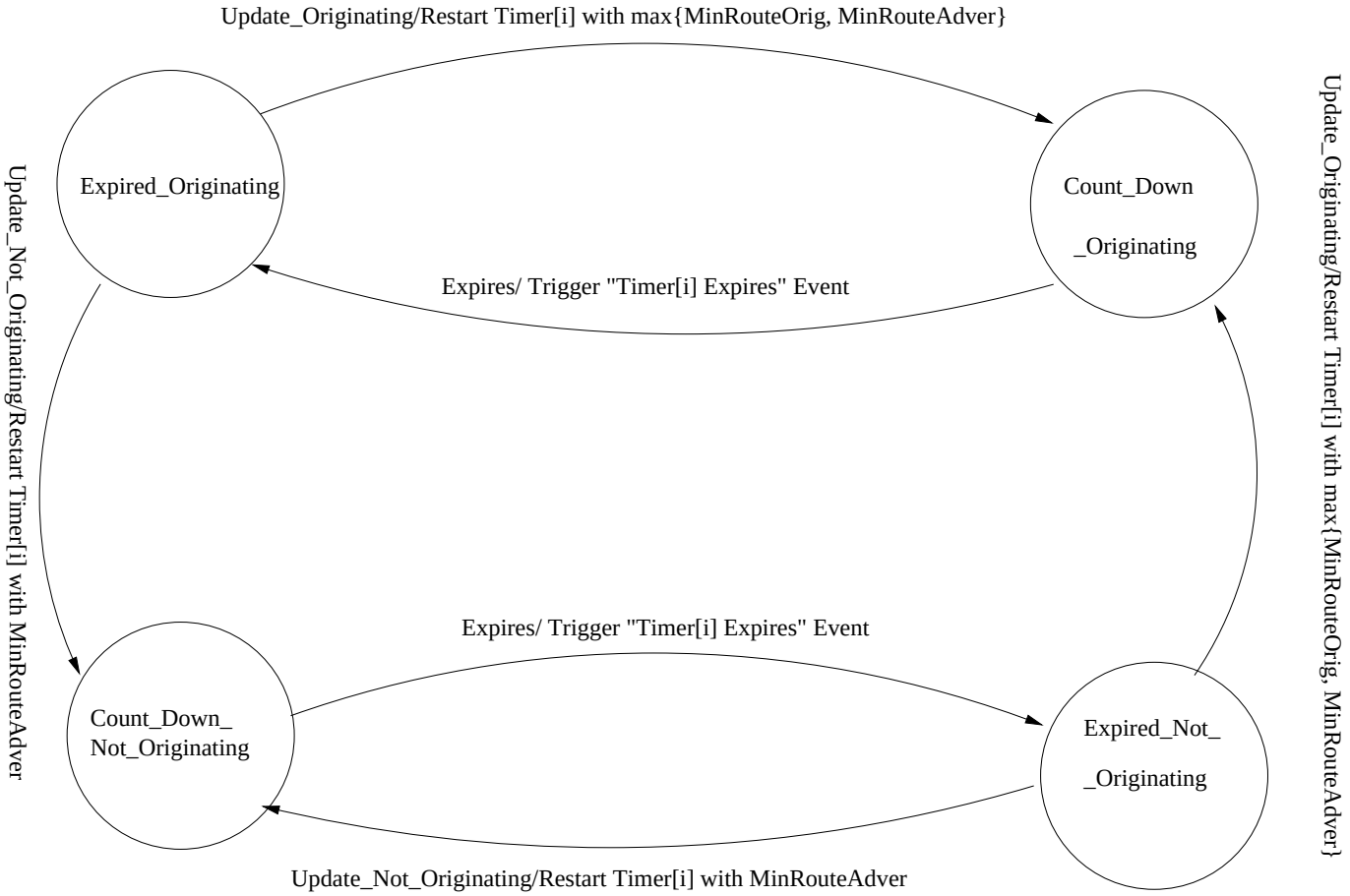


Fig. 7. State Machine for Timer of eBGP Output of prefix p to peer i.

state machine	assertions derived
BGP input (Figure 2)	Assertions 2, 3
iBGP output (Figure 5)	Assertions 1, 3, 4
eBGP output (Figure 6)	Assertions 1, 3, 5, 6

Fig. 8. Assertions

Assertion types:

- MUST: Specification says this assertion is a must.
- IMPLY: Specification implies this assertion, and this assertion is our interpretation.
- SHOULD: Specification says this assertion “should” be followed.
- AMBIGUOUS: Specification is ambiguous on this assertion, or specification should have implied this assertion according to our understanding.

Furthermore, for each assertion, we will investigate their implications by looking at their impact on (some or all of)following three aspects:

- packet forwarding: Whether (and how) packet forwarding will be affected if the assertion is not met.
- routing overhead: Whether failure to meet the assertion will lead to unnecessary routing overhead.

A. Assertion 1: New Information Assertion

[IMPLY]If no refresh is used, each inbound UPDATE from BGP speaker N to BGP speaker R should consist of some new information different from what R already have.

Implications:

a. packet forwarding: not directly affected

b. routing overhead: if an UPDATE with no new information is allowed and it is not processed properly(update is propagated), large amount of BGP update messages could be generated.

c. Out-of-sync of BGP connection will be detected if the assertion fails.

B. Assertion 2: all the latest inbound UPDATE messages should be recorded

[AMBIGUOUS]All the information from the latest inbound UPDATE messages should be stored in RIB-In, even if filtered.

Implications:

a.Usually, packet forwarding is not affected directly. In some rare cases, packet forwarding may be affected. For example, router A receives a route from its neighbor B, and filters it according to the current policy. But later, the policy changes and this route should be valid. If router A does not record the route, it will have no way validate the route, because B will only advertise the route once.

b.Routing overhead is not affected directly.

c.Faulty/malicious withdrawal will be detected if the assertion is met by local BGP speaker.

C. Assertion 3: Withdrawal Assertion

[IMPLY] A BGP speaker should not withdraw a route that it never advertised previously, or a route that was previously advertised but has already been withdrawn.

Implications:

a. Packet forwarding is not directly affected.

b. Unnecessary routing overhead will be led due to propagation of the withdrawals that withdraws the route not previously advertised.

c.Detection of faulty/malicious withdrawal will become easier if the assertion is met by all the BGP speakers.

D. Assertion 4: iBGP propagation Assertion

[MUST] If a new route is learned from external peer and it will be installed in the local BGP speaker's Loc-RIB, this route must be advertised to all other internal peers.

E. Assertion 5: $MinRouteAdver$ Timer Assertion

[MUST] Two consecutive UPDATE messages sent from a single BGP speaker to a specific peer should be separated by at least $MinRouteAdvertisementInterval$ if all of follows are satisfied:

i) the two UPDATE messages are advertising some common set of destinations; and

ii) these two routes are received from EBGP neighbors.

F. Assertion 6: Latest Change Assertion

[SHOULD] Only the last selected route should be advertised at the end of $MinRouteAdvertisementInterval$.

REFERENCES

[1] D. Brand and P. Zafiropulo. On communicating finite-state machines. *J. ACM*, 30(2):323–342, 1983.

[2] C. Labovitz, G. Malan, and F. Jahanian. Internet Routing Instability. In *Proceedings of ACM Sigcomm*, September 1997.

[3] Y. Rekhter, T. Li, and S. Hares. Border Gateway Protocol 4. http://www.ietf.org/internet-drafts/draft-ietf-idr-bgp4-26.txt, Oct 2004.