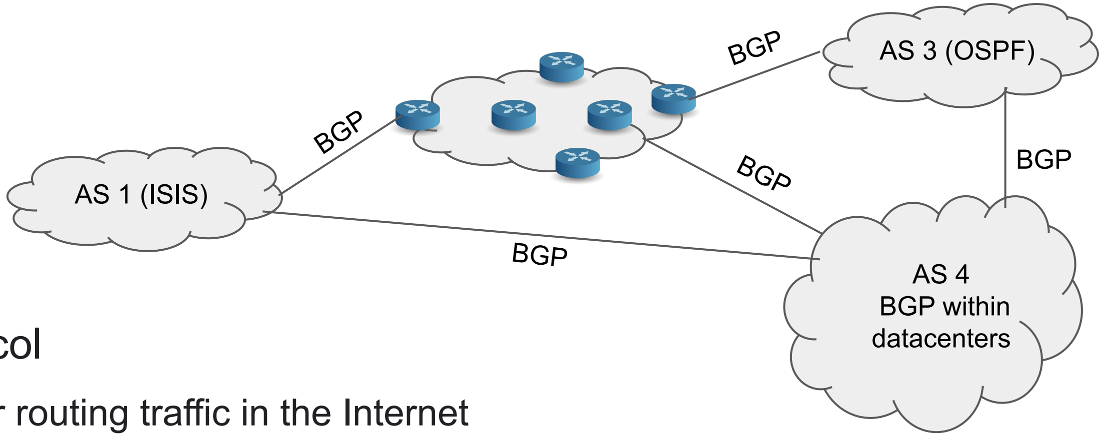
Specifying BGP using TLA+

amanshaikh@google.com

TLA+ community event 2024

2024-09-10

What is BGP?



- BGP = Border Gateway Protocol
  - One of the protocols used for routing traffic in the Internet
- Where is BGP used?
  - Between ASes (Autonomous Systems) comprising the Internet
    - AS: Network administered by a single entity (e.g., company, university, network service provider, cloud service provider, government agency)
      - Example: Google uses AS 15169
  - Within an AS
    - Especially inside datacenters

Why write a TLA+ specification of BGP?

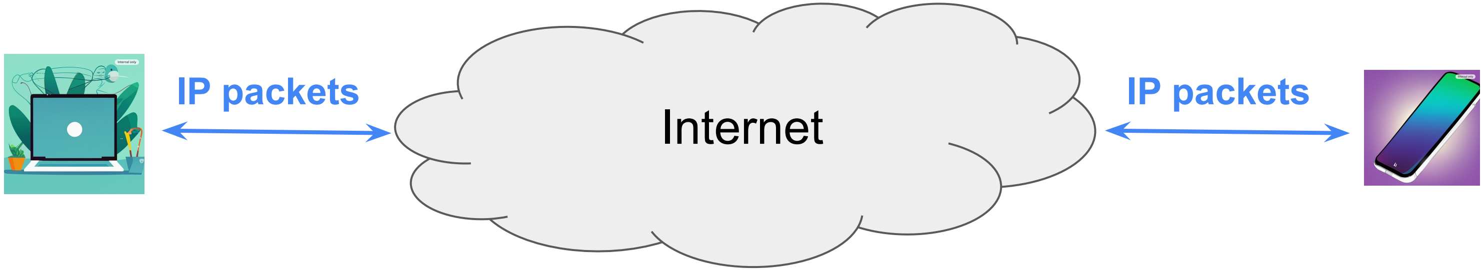
- Explore using TLA+ for formal specification and verification of network designs
  - Focus on the control plane
    - Control plane facilitates communication between network end-points
      - Examples of end-points: servers, desktops, laptops, mobile phones, IoT devices, …
    - Control plane is a complex distributed system
  - BGP is a vital part of the control plane
    - Well-understood, widely used protocol
    - Good starting point for using TLA+ for network verification

Outline

- Background
- TLA+ spec for BGP: overview and key insights
- Concluding remarks
- Appendix: BGP network evolution
  - Variables
  - Initial state and actions
  - Properties

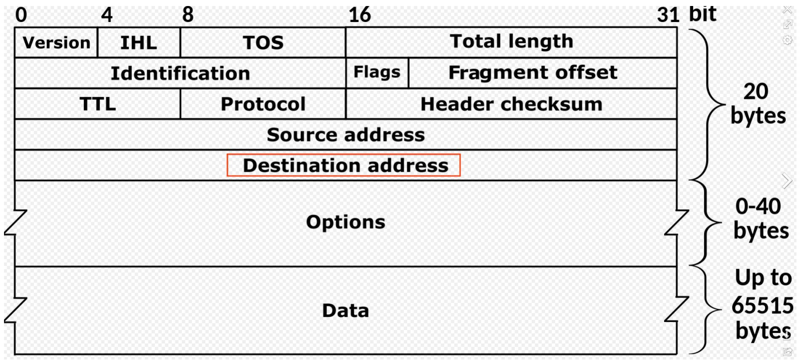
Background

Background: how traffic flows through the Internet

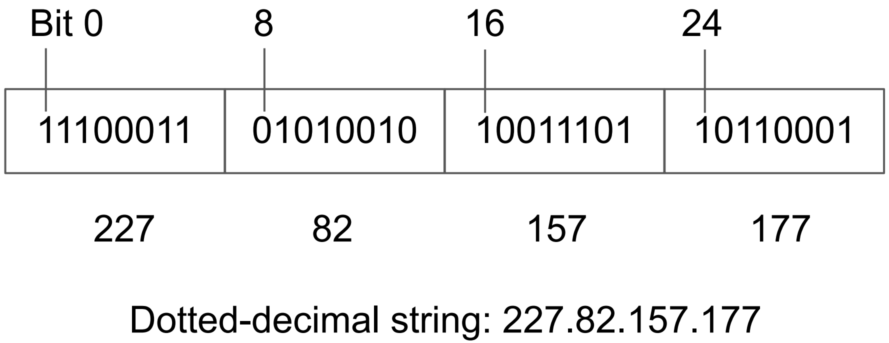


- Internet uses packet switching to enable communication between end-points
  - Applications running on end-points connected to the Internet send messages to one another
    - Examples of messages = Email, chat message, RPC, …
  - Messages are usually divided into IP packets
    - Each packet is forwarded independently (of other packets) through the Internet
    - The path a packet takes from its source to the destination consists of routers

Background: how a router forwards an IP packet



Format of an IPv4 packet (Image source: Internet Protocol version 4 - Wikipedia)



An IPv4 address in its binary form and dotted-decimal format

- A router forwards each packet based on its destination IP address
  - IP address
    - A bit-string of fixed length
    - IPv4 address: 32 bits (usually represented in the dotted-decimal format)
      - Integers in the range [0, 2^{32}-1]

Background: forwarding table in a router

- Forwarding table contains entries against which destination IP addresses are matched
  - Each entry consists of a prefix and outgoing interface of the router
    - An IPv4 prefix is denoted by an address and a mask length (A.B.C.D/M)
      - Mask length corresponds to the number of common bits in IP address
        - Starting from the most significant bit
      - Examples:
        - 0/0 = [0.0.0.0, 255.255.255.255]
        - 10.0.0.0/8 = [10.0.0.0, 10.255.255.255]
        - 10.0.0.0/16 = [10.0.0.0, 10.0.255.255]
        - 11.54.0.0/18 = [11.54.0.0, 11.54.63.255]
      - What if the destination IP address matches more than one prefix?
        - Entry with longest prefix match is used for forwarding the packet
        - Example: 10.0.0.0/16 is used for address 10.0.0.1

Example forwarding table	
Prefix	Outgoing interface
0/0	intf1
10.0.0.0/8	intf2
10.0.0.0/16	intf3
11.54.0.0/18	intf4

- Routing protocols allow a router to populate its forwarding table
  - BGP is one such protocol

Background: how BGP works

- A BGP speaker …
    - establishes sessions with other speakers
    - learns routes over the sessions
        - A route = an IP prefix and other attributes
    - applies policies to every route
        - A policy usually implements business requirements of the AS
    - selects preferred routes for each prefix by comparing attributes of the routes
    - Installs preferred routes in the forwarding table
        - sends preferred routes to other speakers

- BGP speakers determine paths used by packets through route exchange and selection

TLA+ spec for BGP

Overview and insights

TLA+ spec for BGP network evolution: overview

- Initial state:
  - Set of speakers and sessions between speakers
  - Set of routing policies for each session
  - Set of routes
- Actions: speakers select their preferred routes and send them to their neighbors
- What does this spec allow us to do?
  - Check if the network converges to a stable state (i.e. where no action is possible)
  - Inspect what the converged state looks like (i.e. routes selected by each speaker)
- What the spec does not include:
  - Mechanics of BGP session such as TCP and neighbor's BGP state machine
  - Dynamics of the network such as up/down of speakers and sessions

How the spec came together

- Wrote the spec for BGP like I would build a new system from scratch
  - The ‘new’ part = writing TLA+ spec of BGP, not BGP
  - Started with (a module) for specifying IP addresses and prefixes,
    - BGP route selection process
    - BGP route attributes
    - BGP routes
    - …
  - Wrote ‘unit tests’ for modules using TLC
    - Allowed me to understand how TLA+ and TLC works, and gain confidence
    - About 90% lines belong to unit tests
    - **“TLC unit testing was nature’s way of telling me how sloppy my TLA+ modules were”**

The IP address and prefix spec

- IP address = (a non-negative) integer (up to a max value allowed by the number of bits)
- IP prefix = set of integer-ranges such that the range satisfies a property
  - Example prefix: 10.0.0.0/8 = [10.0.0.0, 10.255.255.255] = [167772160, 184549375]
  - Property: the first ‘mask_len’ bits in every integer must be the same
    - E.g., first eight bits must be 0000 1010 for every address covered by 10.0.0.0/8
    - Equivalent properties without bitwise operators in TLA+:
      - The number of addresses in the range must be 2^n for some ‘n’ (n = 24 for 10.0.0.0/8)
      - The range must start at a number divisible by 2^n (for 10.0.0.0/8, 167772160 is divisible by 2^{24})

```
\* The set of IP prefixes.
IpPrefixes == {
  prefix \in IpAddressRanges :
    \E n \in 0..NumOfDigitsForNumberAndBase(MaxIpAddress, 2) :
        \* The size of 'prefix' must be right.
      /\ prefix.high - prefix.low + 1 = 2^n
          \* The 'prefix' must start at the right place.
      /\ prefix.low % 2^n = 0
}
```

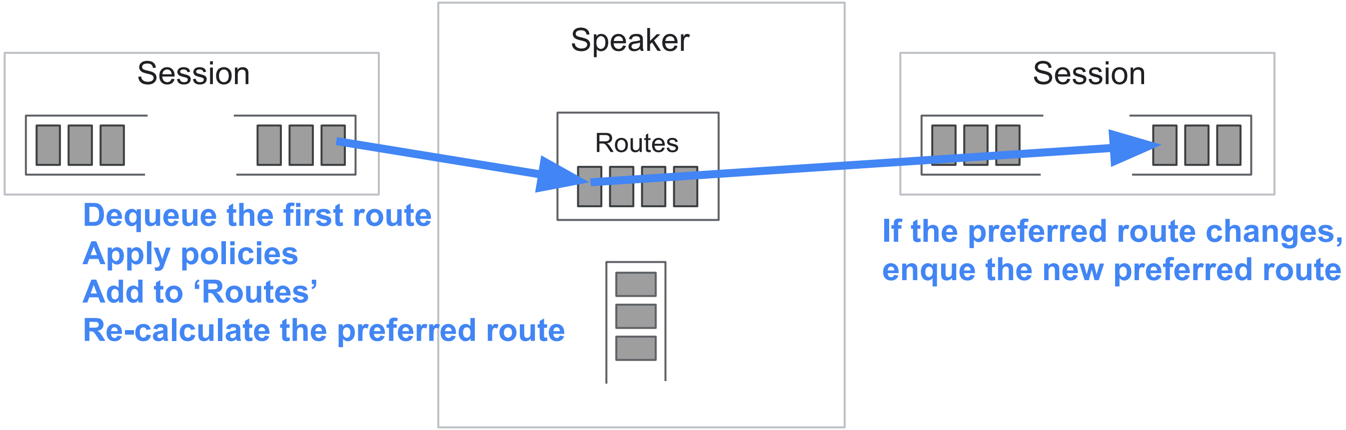
BGP network as a single variable in the spec

- A single variable represents the network of BGP speakers and sessions
  - Network evolves as speakers exchange messages (BGP route updates) over sessions
    - Sessions contain queues where to-be-processed messages are kept

```
\* The network that will undergo evolution.
VARIABLE bgpNetwork
```

- Action occurs when a speaker ‘processes’ a route update
  - Processing a route update
    - Dequeue the route, apply policies, update routing table and enqueue routes for other speakers
  - Single variable allows us to ‘dynamically’ determine which session queues are updated
  - Downside: after every action, TLC prints the entire (updated) network
    - Makes it challenging to know what really changed
- Have reused this pattern in writing specs of a few other network protocols as well

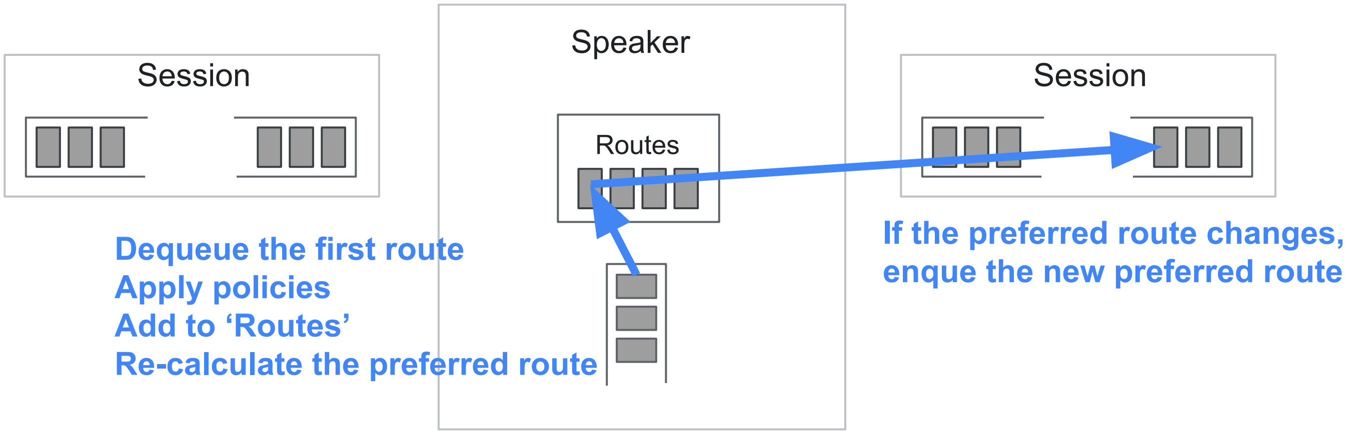
Action: processing a route update enqued at a session




```
\* All possible state-transitions.
Next ==
   \/ \E bgpSession \in bgpNetwork.sessions :
         ServiceSessionIncomingQueue(bgpSession, "AtoZ")
   \/ \E bgpSession \in bgpNetwork.sessions :
         ServiceSessionIncomingQueue(bgpSession, "ZtoA")
   \/ \E bgpSpeaker \in bgpNetwork.speakers :
         ServiceOriginatedRouteUpdateQueue(bgpSpeaker)
   \/ IsNetworkConverged
```

Process a route update
enqued at a session

Action: processing a self-originated route




```
\* All possible state-transitions.
Next ==
  \/ \E bgpSession \in bgpNetwork.sessions :
        ServiceSessionIncomingQueue(bgpSession, "AtoZ")
  \/ \E bgpSession \in bgpNetwork.sessions :
        ServiceSessionIncomingQueue(bgpSession, "ZtoA")
  \/ \E bgpSpeaker \in bgpNetwork.speakers :
        ServiceOriginatedRouteUpdateQueue(bgpSpeaker)
  \/ IsNetworkConverged
```

Process a self-originate route

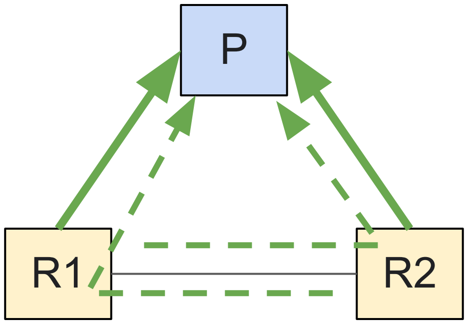
Deadlock is desirable

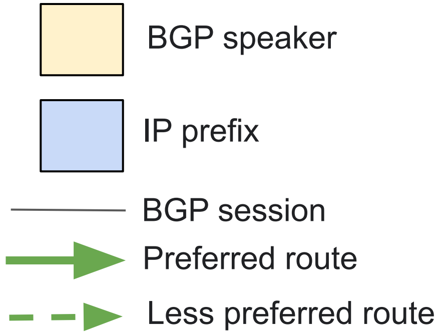
- Unlike a typical TLA+ specification, we want BGP network to deadlock
  - Deadlock means the network has converged
    - Network converges when every speaker is ‘happy’ with its preferred route for every prefix
  - For the spec, convergence happens when no messages remain in session queues

```
\* True if the given 'BgpNetwork' is in a converged state.
IsBgpNetworkConverged(bgpNetwork) ==
  \* For every speaker, there are no more route updates pending in its queue
  \* of self-originated routes.
  /\ \A speaker \in bgpNetwork.speakers:
      speaker.originatedRouteUpdateQueue = <<>>
  \* For every session, there are no route updates pending in the queues
  \* of incoming updates.
  /\ \A session \in bgpNetwork.sessions:
      /\ session.incomingRouteUpdateQueueSpeakerA = <<>>
      /\ session.incomingRouteUpdateQueueSpeakerZ = <<>>
```

- The spec checks certain properties only when network converges
  - Example property: all speakers have routes to every prefix

Example: “good gadget” with two speakers always converges





- Initial state: two BGP speakers with a session between them
  - Each speaker …
    - originates a route to the prefix ‘P’
    - prefers its own route over the route received from the other speaker
- Converged state:
  - Each speaker …
    - learns the route originated by the other speaker
    - continues to prefer the self-originated route

Running TLC on the good gadget

```
{ [ name |-> "R1",
    asn |-> 1,
    routerId |-> 0,
    knownRoutes |->
        { [ nlri |-> [type |-> "string", val |-> [id |-> "P"]],
            routeAttrsSet |->
                { [ resolvableNextHop |-> TRUE,
                    localPref |-> 1,
                    aspath |-> <<>>,
                    origin |-> "igp",
                    med |-> 1,
                    ebgpLearned |-> TRUE,
                    nextHopInteriorCost |-> 1,
                    peerRouterId |-> 0,
                    peerIpAddress |-> 0,
                    locallyOriginated |-> TRUE ],
                  [ resolvableNextHop |-> TRUE,
                    localPref |-> 1,
                    aspath |-> <<2>>,
                    origin |-> "igp",
                    med |-> 1,
                    ebgpLearned |-> TRUE,
                    nextHopInteriorCost |-> 1,
                    peerRouterId |-> 1,
                    peerIpAddress |-> 1,
                    locallyOriginated |-> FALSE ] } ] },
    originatedRouteUpdateQueue |-> <<>> ],
```

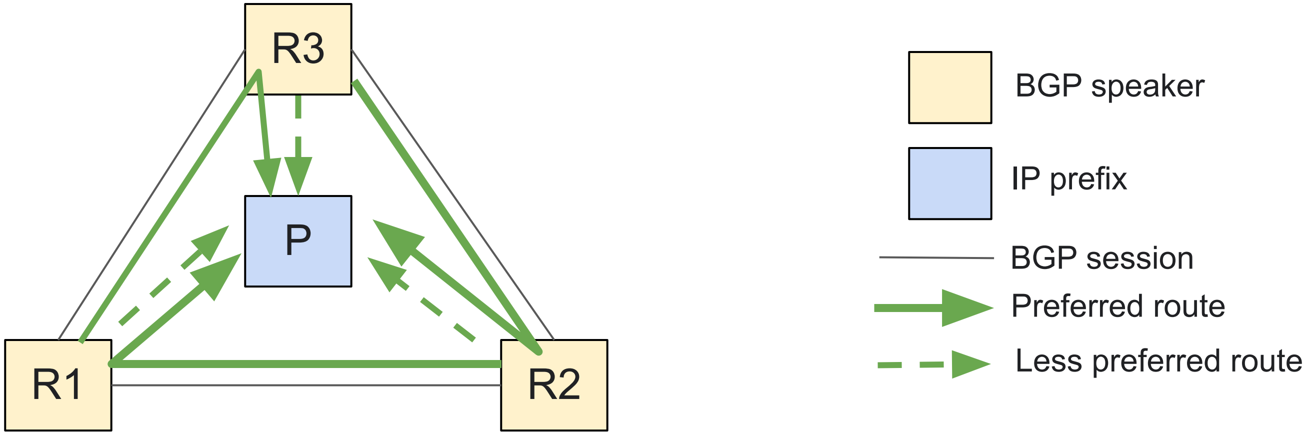
Speaker R1

Prefix P

Self-originated route

Route from speaker R2

Example: “evil gadget” with three speakers never converges



- Initial state: three BGP speakers with pairwise sessions
  - Each speaker …
    - originates a route to the prefix ‘P’
    - prefers route from one of its neighbors over the self-originated route
- Network never converges as …
  - each speaker oscillates between using the self-originated route and the neighbor’s route

Running TLC on the evil gadget

```
+ third_party/java/tlaplus/tlc -config experimental/users/amanshaikh/tla_plus/bgp/examples/BgpNetw
mples/BgpNetworkEvolution_EvilGadgetThreeSpeakers.tla -difftrace -metadir /tmp/states -dfid 25
Error: Invariant BoundedNetworkConvergence is violated.
Error: The behavior up to this point is:
```

“Quick-and-dirty” way of detecting a network that is not likely to ever converge:

Declare error when a network fails to converge after a configurable number of transitions

```
\* The number of transitions 'bgpNetwork' went through.
VARIABLE numTransitions
```

```
\* The number of transitions 'bgpNetwork' can go through without
\* reaching convergence.
BoundedNetworkConvergence ==
  /\ numTransitions <= MaxNumBgpNetworkStates
```

Dealing with state-space explosion

- TLC ran into state-space explosion for networks of moderate size
  - One moderate size network consisted of about ten speakers and ten sessions
  - Main reason:
    - Processing of a single route update can lead to route updates for several other speakers
- Overcoming the state-space explosion
  - Force the spec to process route updates in a deterministic order
    - The order in which sessions can process updates is ‘set’ as part of the initial state
  - Side benefit: quickly check what the converged state looks like and verify its properties

```
\* The initial state.
Init ==
  /\ bgpNetwork = InitialBgpNetwork
  /\ numTransitions = 0
  /\ nameSpeakerSeqForProcessingUpdates = SetToSeq({
       speaker.name : speaker \in InitialBgpNetwork.speakers})
  /\ nameSessionSeqForProcessingUpdates = SetToSeq({
         <<session.nameSpeakerA, session.nameSpeakerZ>> :
           session \in InitialBgpNetwork.sessions
       })
```

Conversion to a sequence forces the spec to process route updates in a deterministic order

Concluding remarks

Summary of the TLA+ spec of BGP

- Represented the network with a single variable
  - Allowed determination of which speakers need to process messages ‘dynamically’
  - Made it challenging to see which part of the network changed in TLC’s state-sequence output
- Deadlock ⇒ convergence of the network
- Dealt with state-space explosion through deterministic order for route update processing

Ongoing work

- Specs for other distributed systems related to network control

- The BGP network evolution spec
    - Extend with more aspects of BGP
    - Release into open source

Experience with TLA+

- First experience with TLA+ and formal methods
  - Learning curve: not too steep
    - Spent some time reading books before writing my first TLA+ module
  - Allowed me to understand the difference between Math and programming
    - Allowed me to fully grasp what functional programming is about

- Model-checking with TLC
  - Allowed unit-testing of TLA+ modules
  - Allowed creation of toy examples of the spec

Appendix: BGP network evolution

Variables

Variables

```
\* The network that will undergo evolution.
VARIABLE bgpNetwork

\* The number of transitions 'bgpNetwork' went through.
VARIABLE numTransitions
```

```
\* The set of BGP networks.
BgpNetworks == {
  bgpNetwork \in [
    speakers: SUBSET(BgpSpeakers),
    sessions: SUBSET(BgpSessions)
  ] : IsBgpNetworkSemanticallyValid(bgpNetwork)
}
```

- Two variables
  - bgpNetwork: a set of speakers and sessions satisfying certain properties
    - Example: at most one session between two speakers
  - numTransitions: the number of TLA+ actions the network undergoes
    - Used for verifying that the network converges within a certain number of actions

Definition of a BGP speaker

```
\* The set of BGP speakers.
BgpSpeakers == {bgpSpeaker \in [
    name: STRING,
    asn: BgpAsns,
    routerId: BgpRouterIds,
    knownRoutes: SUBSET(BgpRoutes),
      \* Set of all routes along with their attributes received by this speaker.
    originatedRouteUpdateQueue: Seq(BgpRouteUpdates)
      \* The queue of yet-to-be-processed BGP route updates originated by this
      \* speaker itself.
  ] :
  \* Every route-attributes in the 'originatedRouteUpdateQueue' uses
  \* 'routerId' as the neighbor IP address, and is 'locallyOriginated'.
  /\ \A originatedRouteUpdate \in
     ValuesOfSeq(bgpSpeaker.originatedRouteUpdateQueue) :
       \A routeAttrs \in originatedRouteUpdate.route.routeAttrsSet :
         /\ routeAttrs.peerIpAddress = bgpSpeaker.routerId
         /\ routeAttrs.locallyOriginated
  \* No two routes in 'knownRoutes' contain the same NLRI.
  /\ Cardinality(bgpSpeaker.knownRoutes) =
     Cardinality({route.nlri : route \in bgpSpeaker.knownRoutes})
}
```

All routes this speaker has received

Self-originated routes that are yet to be processed (i.e., not part of ‘knownRoutes’ of the speaker)

Properties of a speaker

Definition of a BGP session

```
\* The set of BGP sessions.
BgpSessions == {bgpSession \in [
    type: BgpSessionTypes,  \* Type of the session from A to Z direction.
    nameSpeakerA: STRING,
    nameSpeakerZ: STRING,
    asnSpeakerA: BgpAsns,
       \* Usually copied from the ASN of speaker A, but can be different.
    asnSpeakerZ: BgpAsns,
       \* Usually copied from the ASN of speaker Z, but can be different.
    routerIdSpeakerA: BgpRouterIds,
       \* Usually copied from the router-id of speaker A, but can be different.
    routerIdSpeakerZ: BgpRouterIds,
       \* Usually copied from the router-id of speaker Z, but can be different.
    ipAddressSpeakerA: Ipv4Addresses!IpAddresses,
    ipAddressSpeakerZ: Ipv4Addresses!IpAddresses,
        \* Currently we only support IPv4 addresses as session end-point addresses.
    inPolicySpeakerA: BgpPolicies,
        \* Import policies at speaker A for routes coming from speaker Z.
    inPolicySpeakerZ: BgpPolicies,
        \* Import policies at speaker Z for routes coming from speaker A.
    outPolicySpeakerA: BgpPolicies,
        \* Export policies at speaker A for routes going to speaker Z.
    outPolicySpeakerZ: BgpPolicies,
        \* Export policies at speaker Z for routes going to speaker A.
    incomingRouteUpdateQueueSpeakerA: Seq(BgpRouteUpdates),
        \* Sequence of route updates from speaker Z for the post-policy
        \* Adj-RIB-In at speaker A.
    incomingRouteUpdateQueueSpeakerZ: Seq(BgpRouteUpdates)
        \* Sequence of route updates from speaker A for the post-policy
        \* Adj-RIB-In at speaker Z.
  ] :
  IsBgpSessionSemanticallyValid(bgpSession)
}
```

Names of the two speakers

Import and export policies applied in A→Z and Z→A directions

Route updates waiting to be processed by the two speakers

Properties of a session

Properties of a BGP session

```
\* TRUE if 'bgpSession' is syntactically and semantically valid; FALSE otherwise.
LOCAL IsBgpSessionSemanticallyValid(bgpSession) ==
  /\ bgpSession.nameSpeakerA # bgpSession.nameSpeakerZ
  /\ bgpSession.routerIdSpeakerA # bgpSession.routerIdSpeakerZ
  /\ bgpSession.ipAddressSpeakerA # bgpSession.ipAddressSpeakerZ
  \* Route reflector - client session can only be an iBGP sessions.
  \* Note: the reverse is not true as an iBGP session can also be of
  \* type 'ptop'.
  /\ (bgpSession.type = "rtoc" \/ bgpSession.type = "ctor") =>
      bgpSession.asnSpeakerA = bgpSession.asnSpeakerZ
  \* All route-attributes in the queues of incoming route updates
  \* do not have 'locallyOriginated' set.
  /\ \A routeUpdate \in
     ValuesOfSeq(bgpSession.incomingRouteUpdateQueueSpeakerA) :
        \A routeAttrs \in routeUpdate.route.routeAttrsSet :
           /\ routeAttrs.locallyOriginated = FALSE
  /\ \A routeUpdate \in
     ValuesOfSeq(bgpSession.incomingRouteUpdateQueueSpeakerZ) :
        \A routeAttrs \in routeUpdate.route.routeAttrsSet :
           /\ routeAttrs.locallyOriginated = FALSE
```

Properties of a BGP network

```
\* TRUE if a given 'bgpNetwork' is semantically valid; FALSE otherwise.
LOCAL IsBgpNetworkSemanticallyValid(bgpNetwork) ==
  \* Every speaker has a distinct name.
  /\ Cardinality(bgpNetwork.speakers) =
      Cardinality({speaker.name : speaker \in bgpNetwork.speakers})
  \* Every speaker has a distinct router-id.
  /\ Cardinality(bgpNetwork.speakers) =
      Cardinality({speaker.routerId : speaker \in bgpNetwork.speakers})
  \* Both ends of every session are speakers of the network.
  /\ \A session \in bgpNetwork.sessions :
        /\ session.nameSpeakerA \in
            {speaker.name : speaker \in bgpNetwork.speakers}
        /\ session.nameSpeakerZ \in
            {speaker.name : speaker \in bgpNetwork.speakers}
  \* If a session from A 'to' Z is included in sessions =>
  \*    a session from Z 'to' A is NOT included.
  /\ \A sessionAtoZ \in bgpNetwork.sessions :
        ~\E sessionZtoA \in bgpNetwork.sessions :
          /\ sessionAtoZ.nameSpeakerA = sessionZtoA.nameSpeakerZ
          /\ sessionAtoZ.nameSpeakerZ = sessionZtoA.nameSpeakerA
  \* Every session is between a distinct pair of IP addresses.
  /\ Cardinality(bgpNetwork.sessions) =
      Cardinality({<<session.ipAddressSpeakerA, session.ipAddressSpeakerZ>> :
                  session \in bgpNetwork.sessions})
```

Appendix: BGP network evolution

Initial state and actions

Initial state and actions

- Initial state:

```
Init ==
  /\ bgpNetwork = InitialBgpNetwork
  /\ numTransitions = 0
```

InitialBgpNetwork is a constant specified in .cfg file

- All possible actions:

```
\* All possible state-transitions.
Next ==
  \/ \E bgpSession \in bgpNetwork.sessions :
        ServiceSessionIncomingQueue(bgpSession, "AtoZ")
  \/ \E bgpSession \in bgpNetwork.sessions :
        ServiceSessionIncomingQueue(bgpSession, "ZtoA")
  \/ \E bgpSpeaker \in bgpNetwork.speakers :
        ServiceOriginatedRouteUpdateQueue(bgpSpeaker)
  \/ IsNetworkConverged
```

Process A→Z route update

Process Z→A route update

Process self-originated update

Network converges

  - Processing a route at a speaker:
    - Apply import policies
    - Run route selection process
    - Enque new preferred route in the session queues (of neighbors) if preferred route changes
  - Network is converged if all queues are empty
    - Equivalent to the network becoming deadlocked
    - Represented as an action to allow checking certain properties upon convergence

Actions that result in processing of a route update

```
\* Action when a queue of the given 'bgpSession' contains at least one
\* route update.
ServiceSessionIncomingQueue(bgpSession, bgpRoutePropagationDirection) ==
  LET queueName ==
    BgpSessionIncomingRouteUpdateQueueForRoutePropagationDirection(
      bgpRoutePropagationDirection)
  IN
    /\ bgpSession[queueName] # <<>>
    /\ bgpNetwork' =
         BgpNetworkAfterProcessingRouteUpdateFromSessonIncomingQueue(bgpNetwork,
         bgpSession.nameSpeakerA, bgpSession.nameSpeakerZ,
         bgpRoutePropagationDirection, BgpRouteSelectionOptionsVal)
    /\ numTransitions' = numTransitions + 1
```

- Select between A→Z or Z→A direction of the session
- The chosen queue must have at least one route update
- Updated ‘bgpNetwork’ after the route update is ‘processed’

```
\* Action when the queue of self-originated route updates of the given
\* 'bgpSpeaker' has a route update.
ServiceOriginatedRouteUpdateQueue(bgpSpeaker) ==
  /\ bgpSpeaker.originatedRouteUpdateQueue # <<>>
  /\ bgpNetwork' =
     BgpNetworkAfterProcessingRouteUpdateFromSpeakerOriginatedQueue(bgpNetwork,
       bgpSpeaker.name, BgpRouteSelectionOptionsVal)
  /\ numTransitions' = numTransitions + 1
```

- The queue at the speaker must have at least one route update
- Updated ‘bgpNetwork’ after the route update is ‘processed’

Processing a route update from a session queue

```
\* BGP network resulting after processing a route update from the front
\* of a session's queue of incoming updates.
\* NOTE: there should be at least one route update in the queue.
BgpNetworkAfterProcessingRouteUpdateFromSessonIncomingQueue(bgpNetwork,
  nameSpeakerA, nameSpeakerZ, bgpRoutePropagationDirection,
  bgpRouteSelectionOptions) ==
  LET
    bgpSession ==
      BgpSessionWithGivenNamesInBgpNetwork(bgpNetwork, nameSpeakerA, nameSpeakerZ)
    routeUpdate ==
      BgpRouteUpdateAtFrontOfSessionIncomingQueue(bgpNetwork, nameSpeakerA,
        nameSpeakerZ, bgpRoutePropagationDirection)
    bgpNetworkAfterRemovingRouteUpdate ==
      BgpNetworkAfterRemovingRouteUpdateFromSessionIncomingQueue(bgpNetwork,
        nameSpeakerA, nameSpeakerZ, bgpRoutePropagationDirection)
    nameSpeaker ==
      IF bgpRoutePropagationDirection = "AtoZ" THEN
        nameSpeakerZ
      ELSE
        nameSpeakerA
  IN
    BgpNetworkAfterProcessingRouteUpdateBySpeakerAndSessions(
      bgpNetworkAfterRemovingRouteUpdate, nameSpeaker, routeUpdate,
      bgpRouteSelectionOptions)
```

Remove the update from the front of the queue

Process the update

A step in processing a route update: route selection

```
\* The set of BGP route attributes from 'routeAttrsSet' that are best according
\* to the selection process with 'routeSelectionOptions'.
BgpRouteAttrsSetWithBestAttrs(routeAttrsSet, routeSelectionOptions) ==
  IF routeAttrsSet = {} THEN {}
  ELSE
    LET
      routeAttrsSet1 == BgpRouteAttrsSetWithResolvableNextHop(routeAttrsSet)
      routeAttrsSet2 == BgpRouteAttrsSetWithBestLocalPref(routeAttrsSet1)
      routeAttrsSet3 == BgpRouteAttrsSetWithBestAspathLength(routeAttrsSet2)
      routeAttrsSet4 == BgpRouteAttrsSetWithBestOrigin(routeAttrsSet3)
      routeAttrsSet5 ==
        IF routeSelectionOptions.compareMedPerNeighborAsn THEN
          BgpRouteAttrsSetWithBestMedsPerNeighborAsns(routeAttrsSet4)
        ELSE
          BgpRouteAttrsSetWithBestMed(routeAttrsSet4)
      routeAttrsSet6 == BgpRouteAttrsSetWithEbgpLearnedPreference(routeAttrsSet5)
      routeAttrsSet7 == BgpRouteAttrsSetWithBestNextHopInteriorCost(routeAttrsSet6)
      routeAttrsSet8 == BgpRouteAttrsSetWithBestPeerRouterId(routeAttrsSet7)
      routeAttrsSet9 == BgpRouteAttrsSetWithBestPeerIpAddress(routeAttrsSet8)
    IN
      routeAttrsSet9
```

- Route selection (at each speaker)
  - Essentially a lexicographic ordering of routes based on various attributes
  - Performed independently for each prefix

Definition of a BGP route and a route update

```
\* The set of BGP routes.
BgpRoutes == [
  nlri: BgpNlris,
  \* Upto 'BgpMaxNumOfAttrsSetsInRoute' BGP route attributes.
  routeAttrsSet: MaxSizeSet(BgpRouteAttrsSet, BgpMaxNumOfAttrsSetsInRoute)
]
```

NLRI is a generic term for an IP prefix

Set of attributes

```
\* The set of BGP route updates.
BgpRouteUpdates == [
  type: BgpRouteUpdateTypes,
  senderIpAddress: BgpNeighborAddresses,
  route: BgpRoutes
]
```

Type: whether the route should be added/updated or removed

Sender of the route

The route

BGP policies

```
\* The set of BGP policies. Each policy is a function that converts a 'BgpRoute'
\* to another 'BgpRoute' or 'BgpNonRoute', and 'BgpNonRoute' to 'BgpNonRoute'.
BgpPolicies ==
  {policy \in
   [BgpRoutes \union {BgpNonRoute} -> BgpRoutes \union {BgpNonRoute}] :
    policy[BgpNonRoute] = BgpNonRoute}
```

- A BGP policy is a function that maps a BGP route to (another) route
  - Usually modifies one or more attributes of a route
  - ‘BgpNonRoute’ allows a speaker to drop a route
    - Examples:
      - A speaker decides not to accept an incoming route
      - A speaker decides not to send a route to some peer

Appendix: BGP network evolution

Properties

Properties

All properties checked

```
\* The specification.
Spec ==
  /\ Init
  /\ [][Next]_vars
  /\ []TypeInvariant /\ []BoundedNetworkConvergence /\ []ConvergenceInvariant
  /\ NetworkConvergesEventually
```

Type-checking

Invariant to be checked only upon convergence

Network converges within a certain number of actions

Property checked once the network converges

```
ConvergenceInvariant ==
  (/\ ShouldCheckReachabilityUponConvergence
   /\ ENABLED IsNetworkConverged) =>
      CanAllSpeakersReachAllNlrisInBgpNetwork(bgpNetwork,
        BgpRouteSelectionOptionsVal)
```

Once the network has converged, check reachability from all speakers to all NLRIs