Network Service Delivery and Throughput Optimization Via Software Defined Networking

Thesis Defense:

Aaron Rosen

Committee:

Kuang-Ching “KC” Wang
Harlan Russell
Sebastien Goasguen

Holcombe Department of Electrical & Computer Engineering

Overview

- **Network Service Delivery and Throughput Optimization Via Software Defined Networking**
  - Software Defined Networking (SDN)
  - Transmission Control Protocol (TCP)
  - Steroid OpenFlow Service (SOS)
  - GENI
  - Local Test bed

Software Defined Networking

- Network architecture which decouples control plane and forwarding plane of network devices
- Allows network devices to be programed via software API
- This allows new ideas to be quickly prototyped and tested at large scale
- OpenFlow is one example of SDN

The OpenFlow SDN Approach

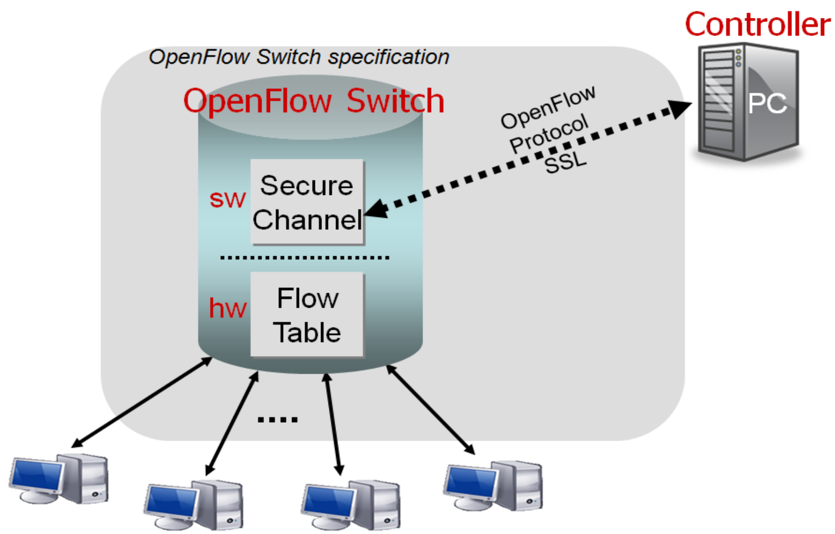


Illustration Courtesy of Stanford University Clean-Slate Program

- Switches connect to remote controllers
- Controllers install flow entries into switches to handle packets
- Packets not matching any flow entry are sent to the controller (via Packet_in)
- Controller decides how to handle packet
- Allows traffic to be manipulated easily

OpenFlow Tables

Rule	Action	Stats

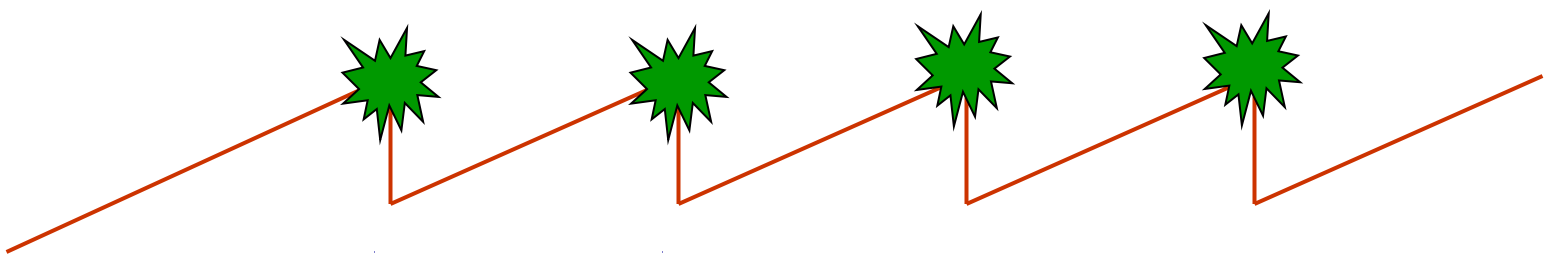
Packet + byte counters

1. Forward packet to port(s)
2. Drop packet
3. Packet header rewrite

Switch Port	MAC src	MAC dst	Eth type	VLAN ID	IP Src	IP Dst	IP Prot	TCP sport	TCP dport

Transmission Control Protocol (TCP)

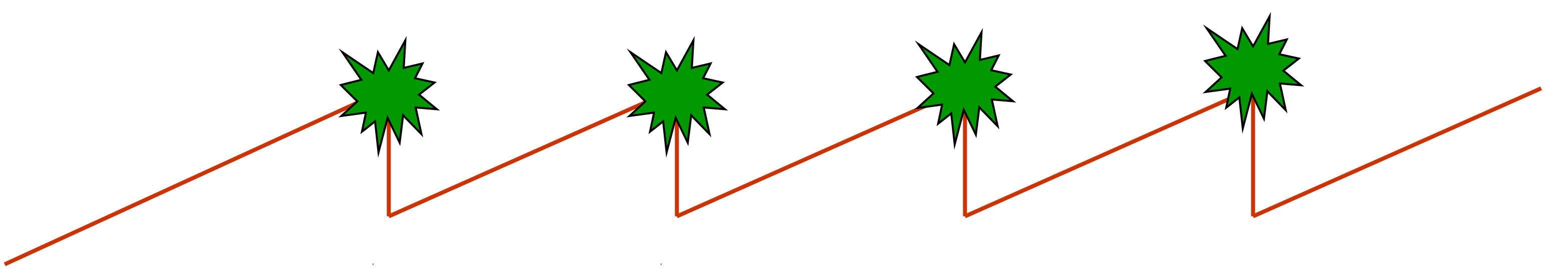
- De facto protocol for transmitting data reliably over the internet.
- Uses congestion control algorithms
- Performance degrades heavily with packet loss and high latency.
- High latency results in a higher Bandwidth Delay Product requiring a large buffer to store packets in flight before they are acknowledged



Transmission Control Protocol (TCP)

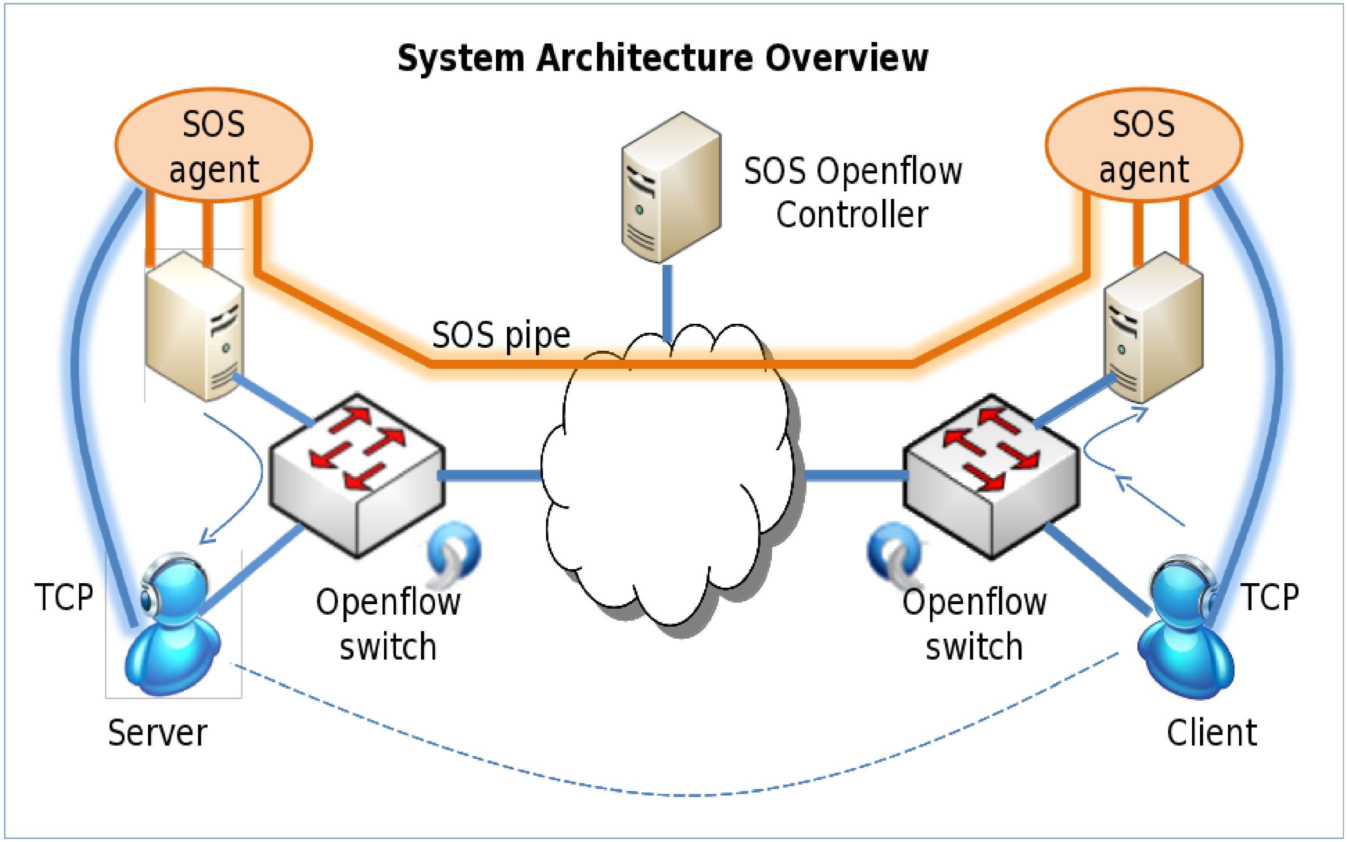
- Lots of past work has investigated and proposed solutions to these issues.
- Unfortunately, most of these solutions require modifications to end users machines.
- GridFTP is one example that is used to move terabytes of data across the globe each year generated by the Large Hadron Collider.
- Requiring specialized software adds complexity and cost for end users.

- Can these complexities be removed with SDN?



Steroid OpenFlow Service

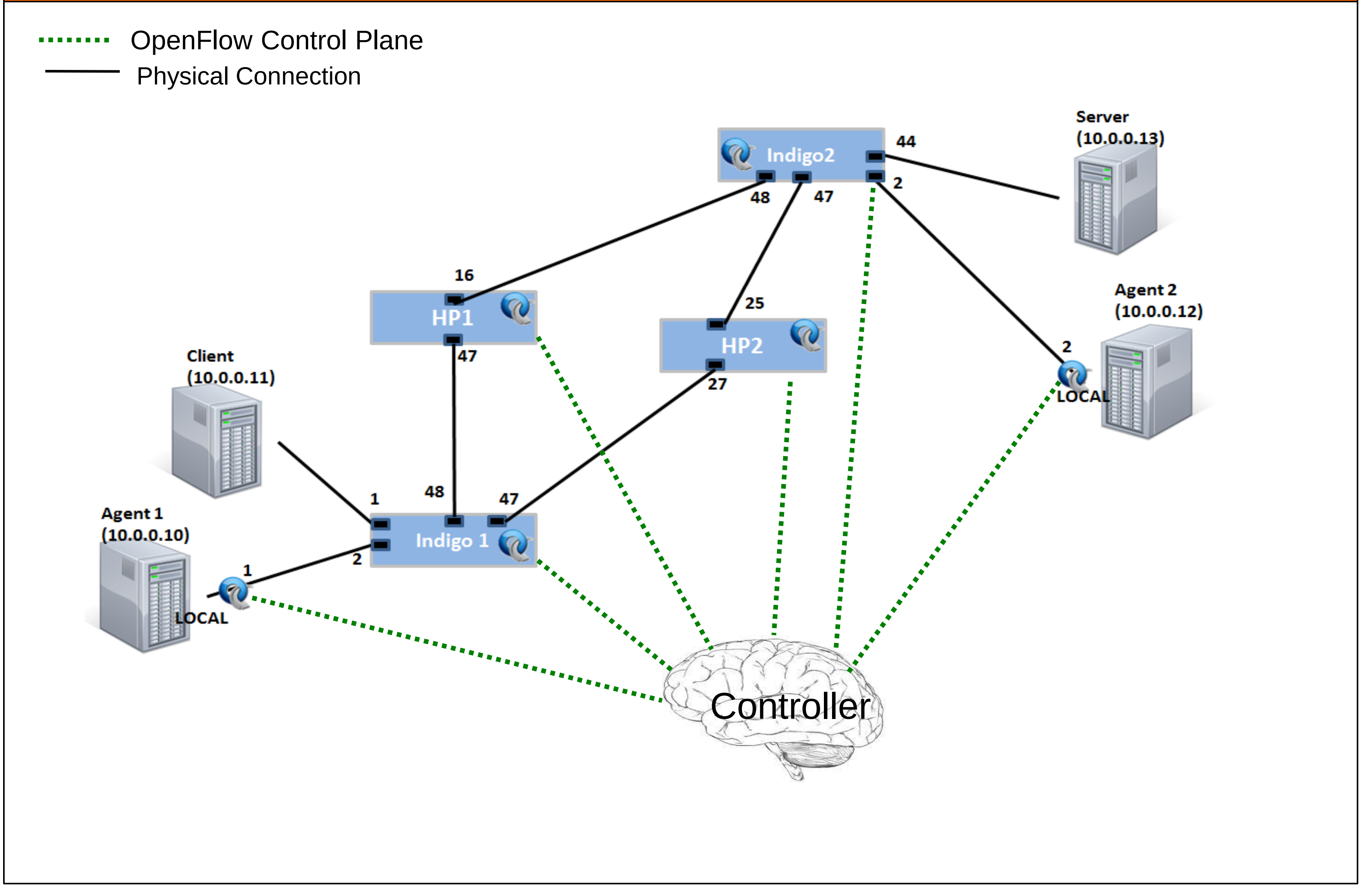
- SOS provides a seamless enhancement to end-to-end application throughput over long range networks.
- Decouples users protocol from network



SOS Overview

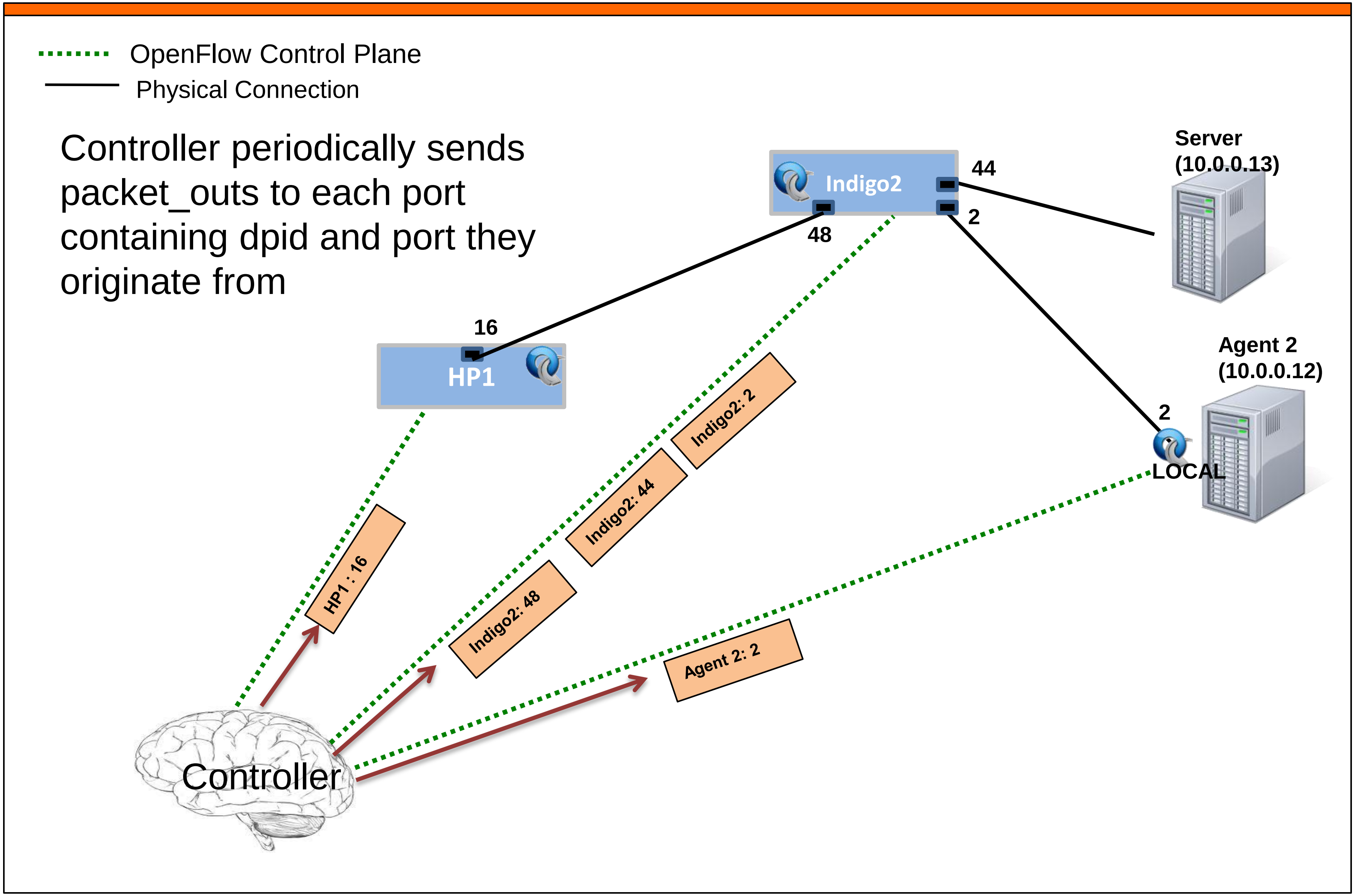
- Goal: no modifications to host, seamless improvement
- Solution:
  - OpenFlow network detects TCP connection (client-server)
  - OpenFlow network redirects connection to local SOS agent
  - SOS agent starts high throughput transport to SOS agent on destination site
  - Destination SOS agent starts TCP connection to server
  - OpenFlow network discovers all sites with SOS agents
  - OpenFlow network allows multiple path transport

Example Topology

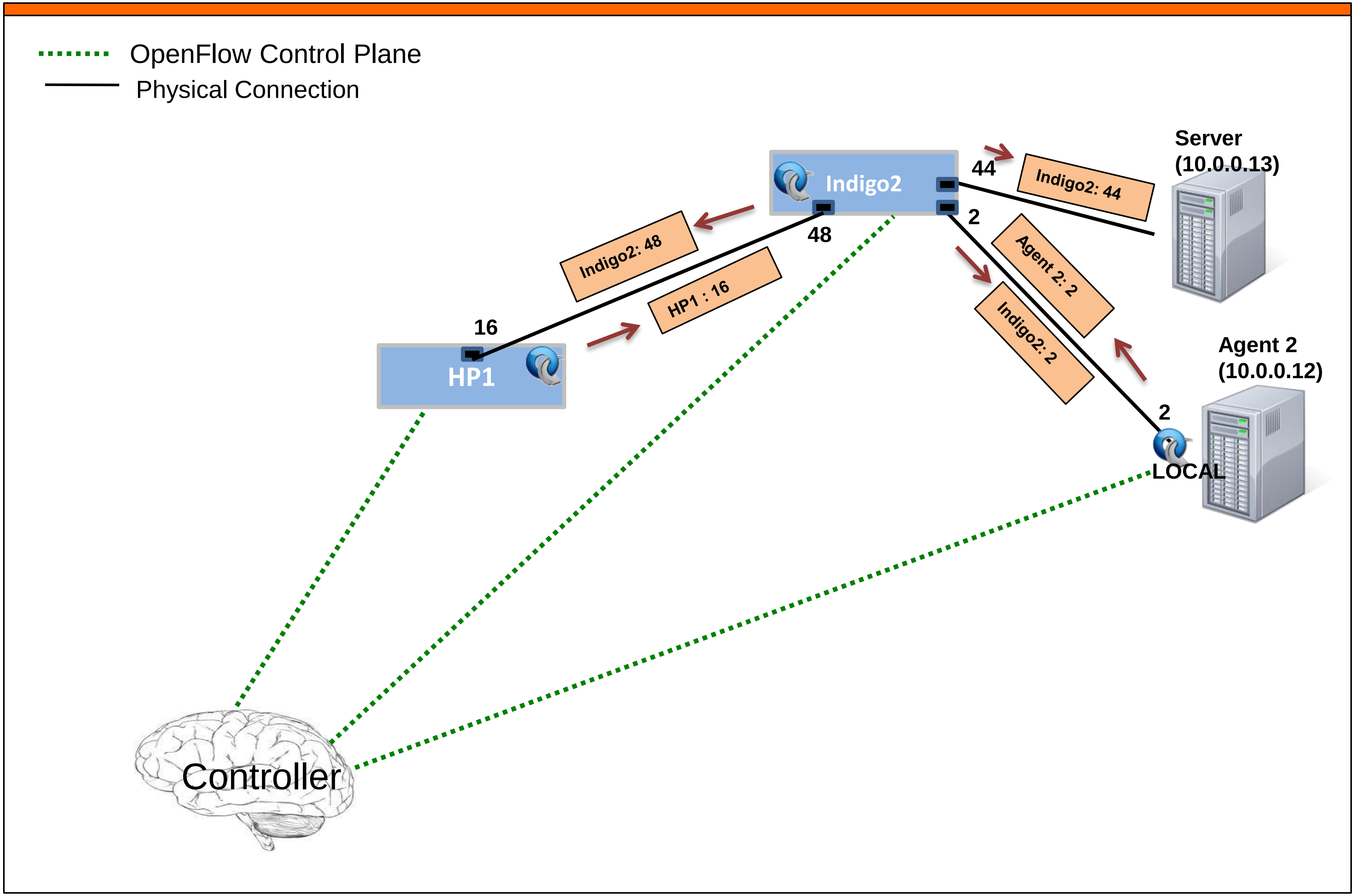


Datapath Topology Discovery Controller Sends Packet_out

Controller periodically sends packet_outs to each port containing dpid and port they originate from

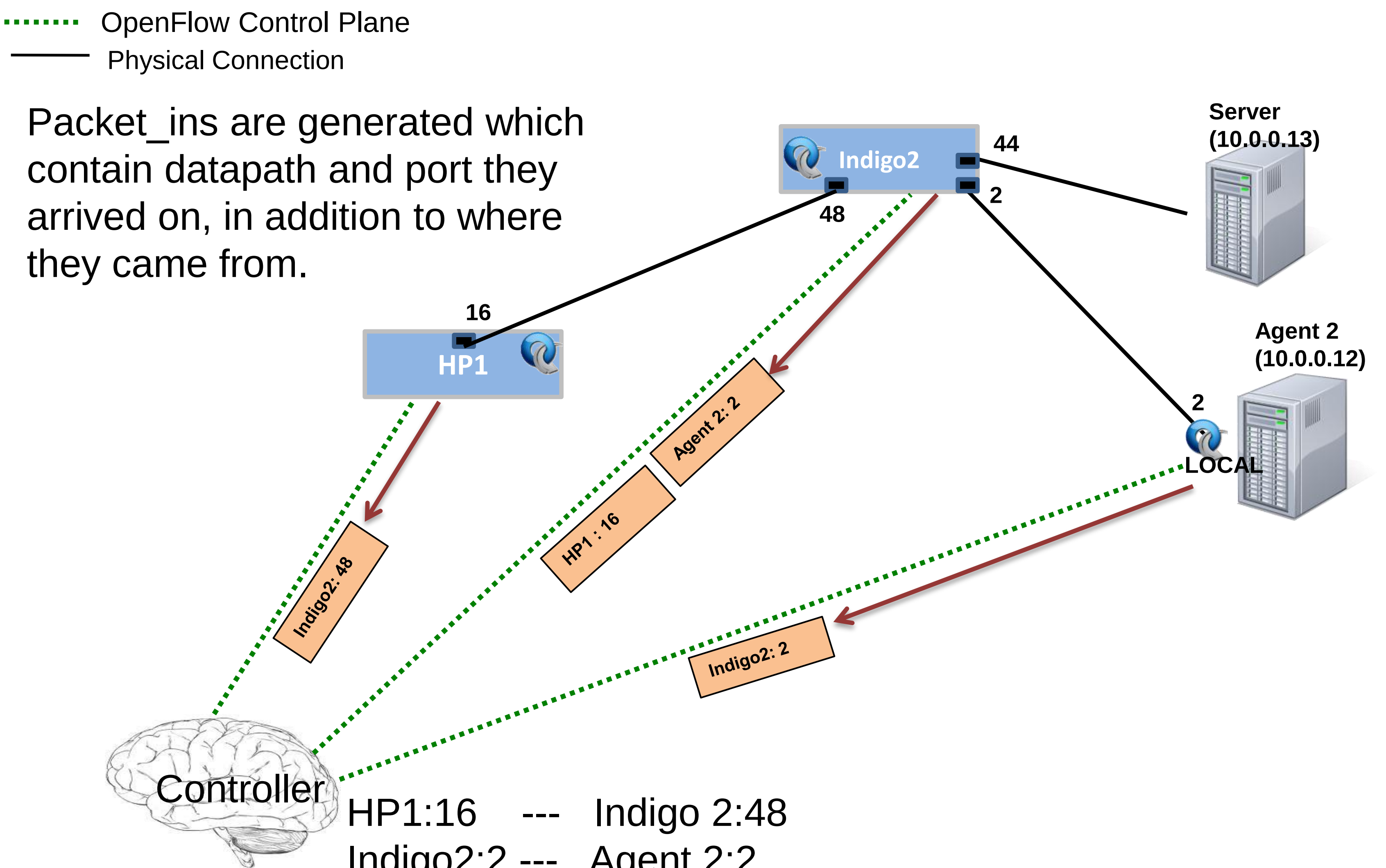


Discovery packets leave desired interface



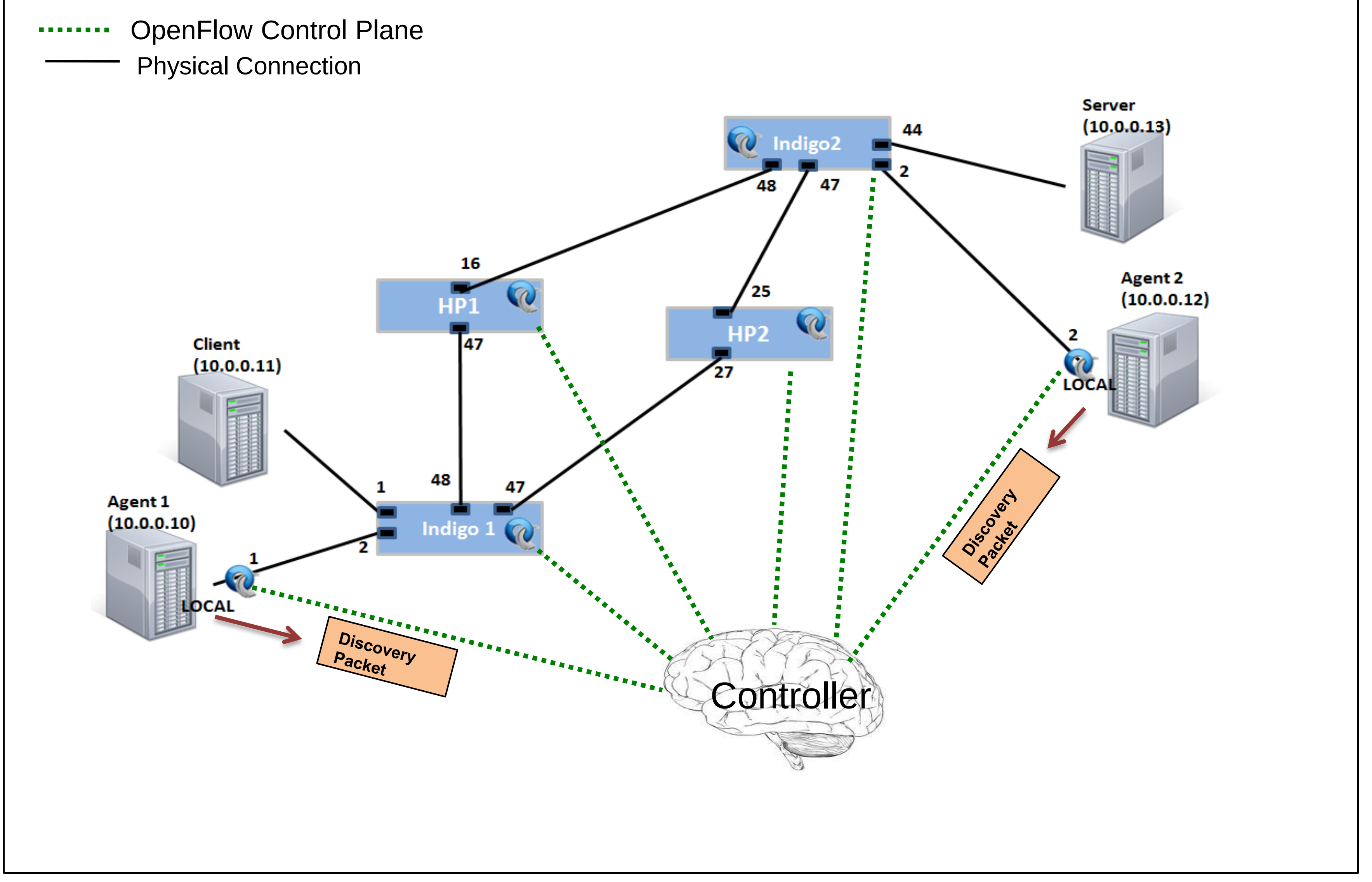
Packets are returned to controller via Packet_in

Packet_ins are generated which contain datapath and port they arrived on, in addition to where they came from.

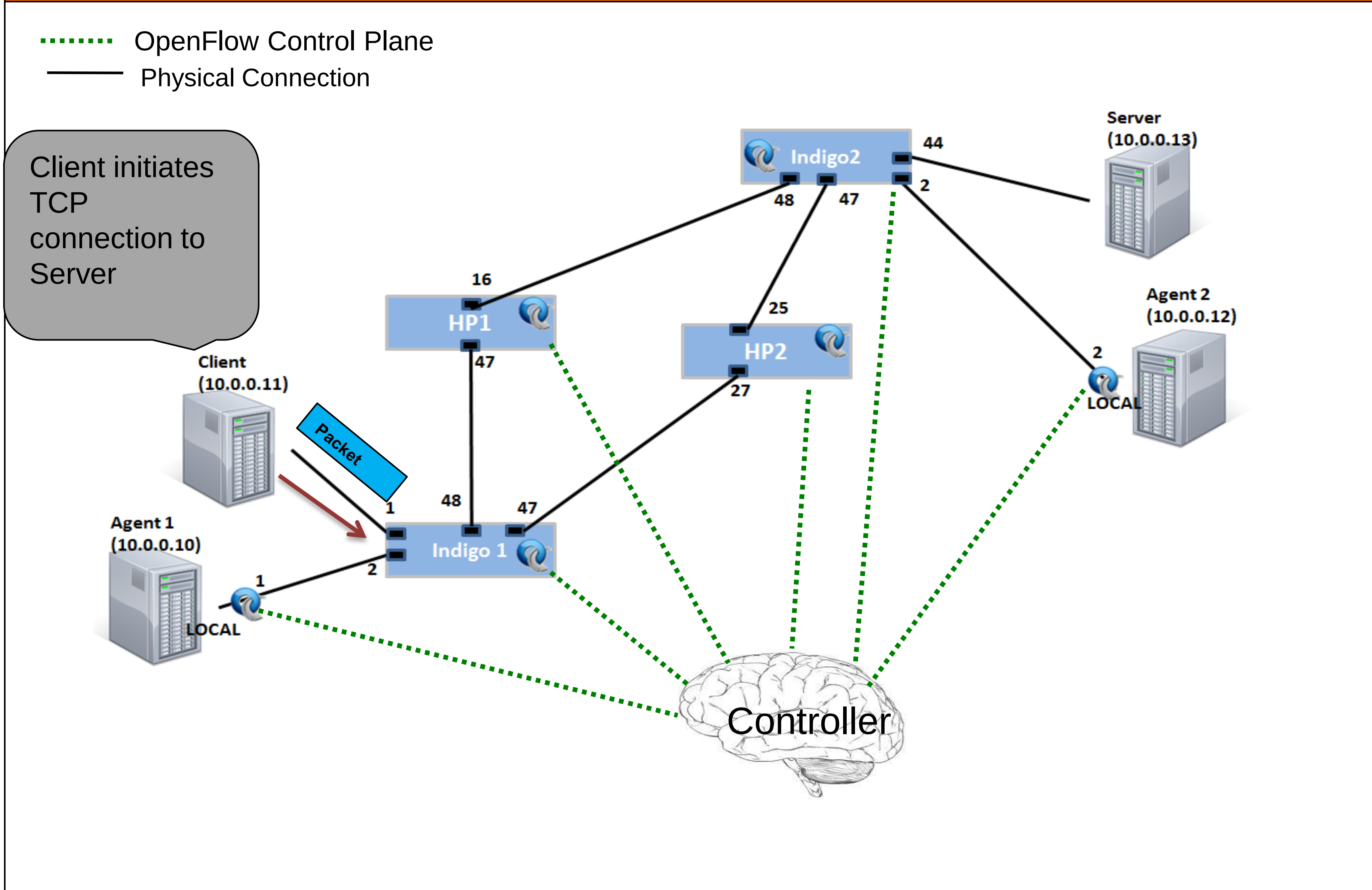


HP1:16 --- Indigo 2:48
Indigo2:2 --- Agent 2:2

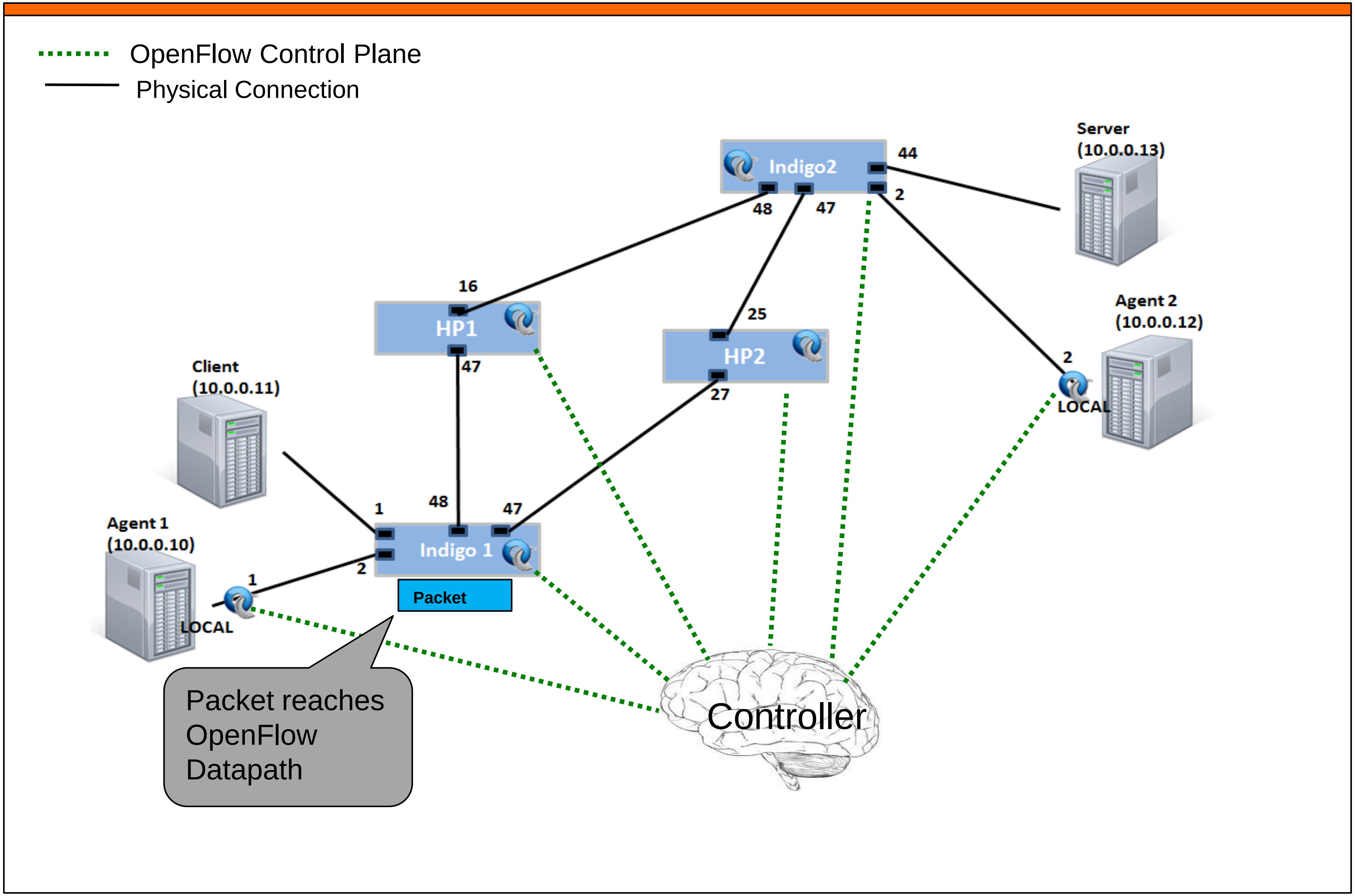
Agent Discovery



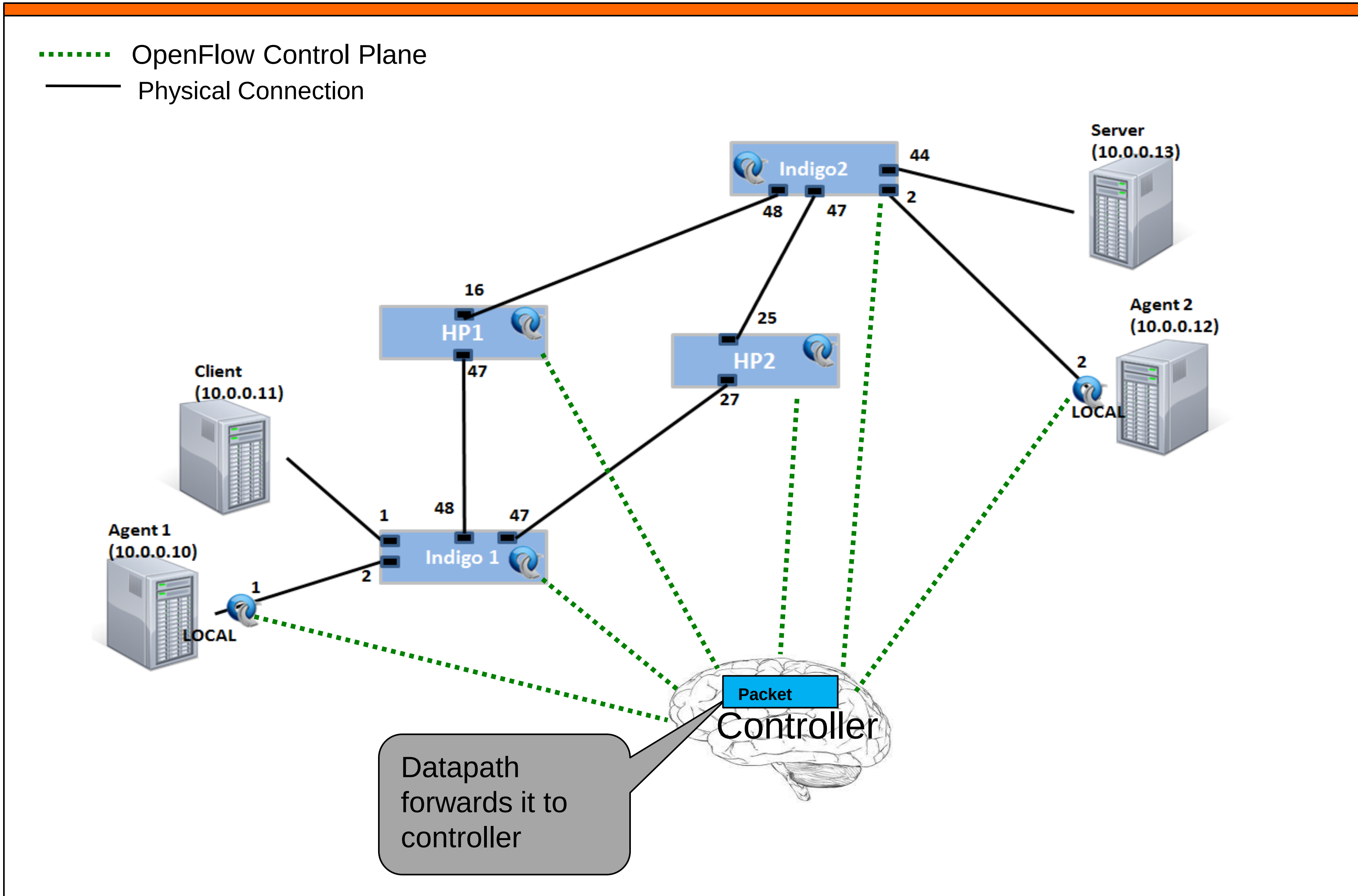
SOS: Client Initiates TCP Connection



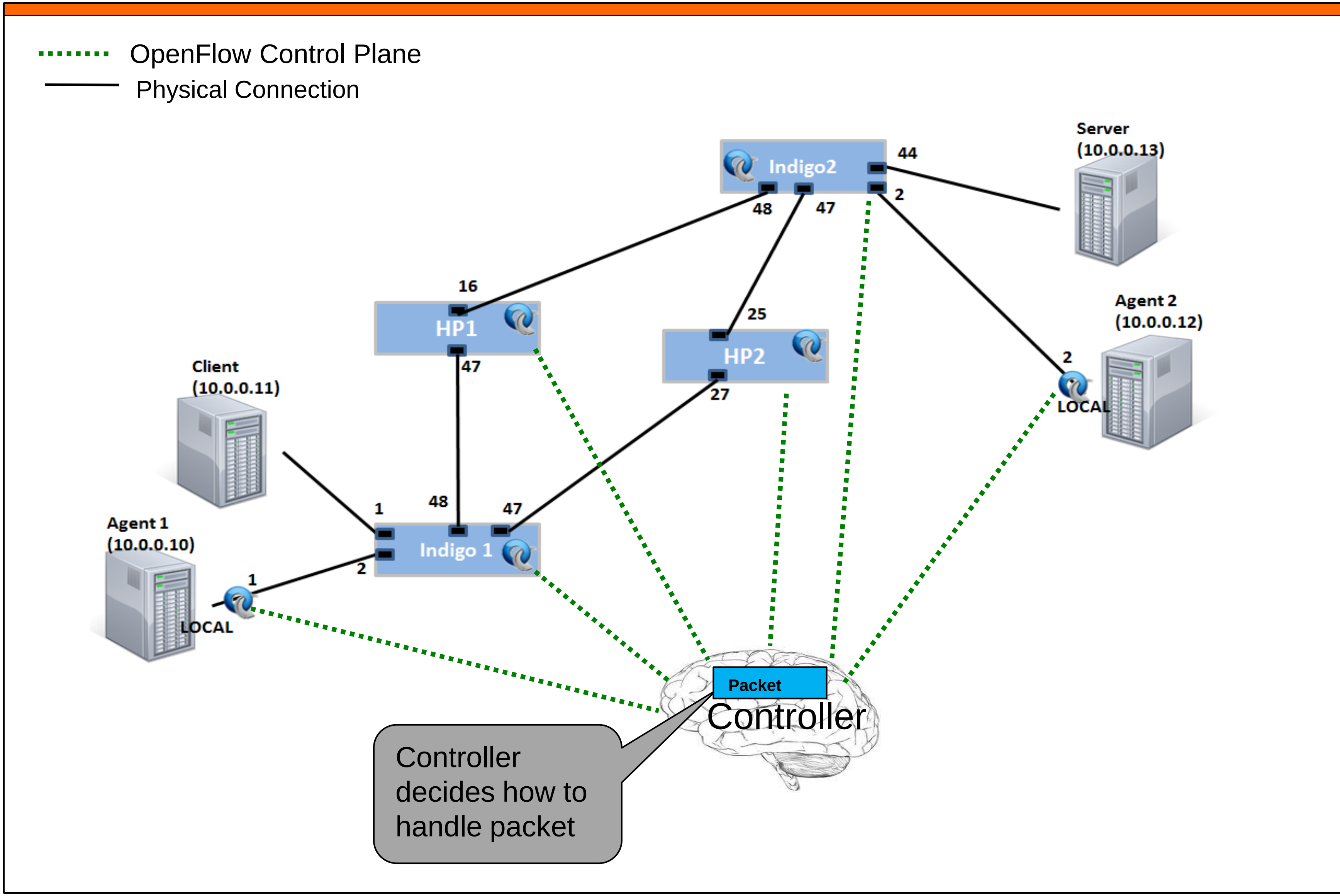
Packet Reaches First OpenFlow Datapath



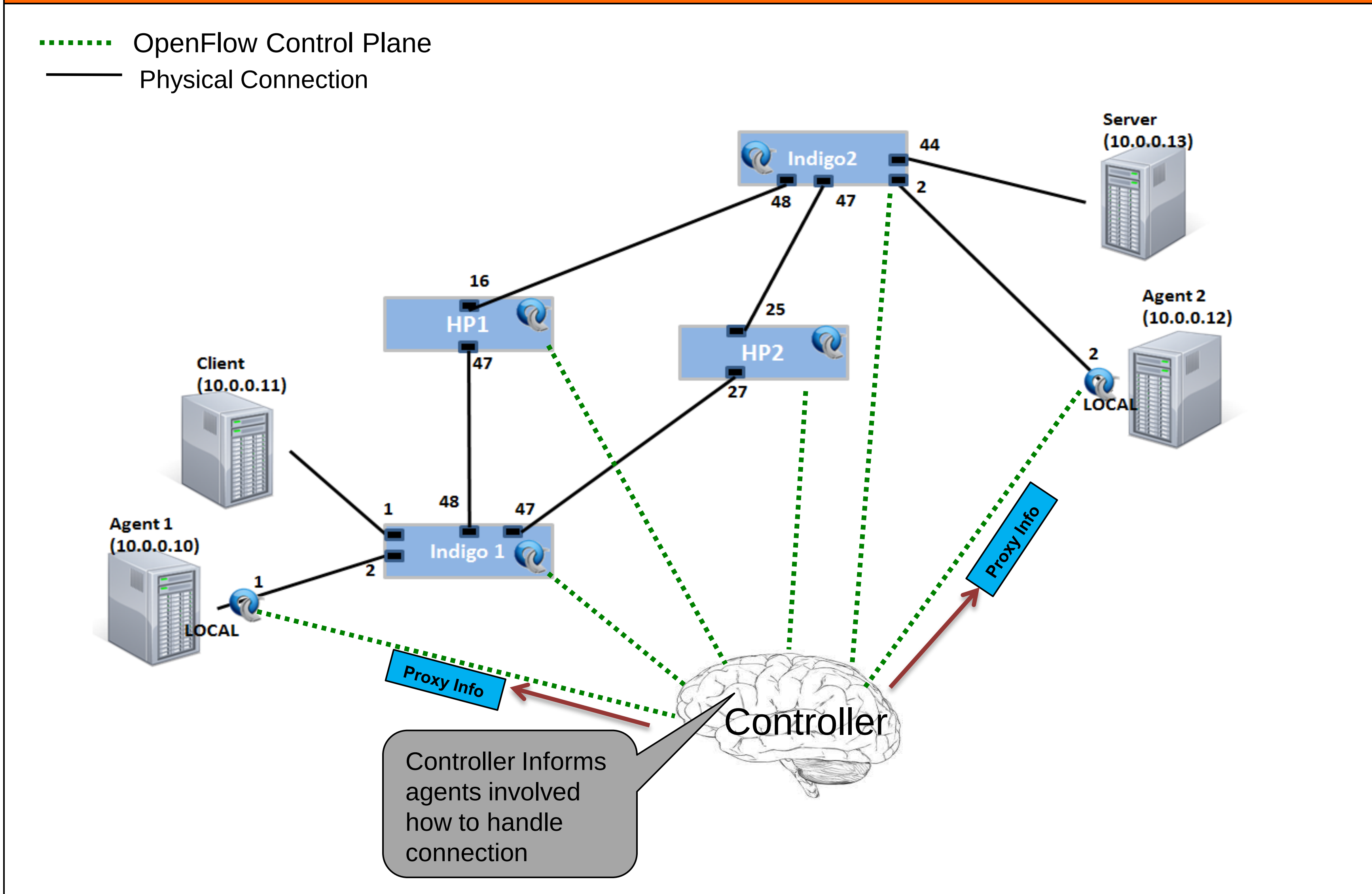
Packet Forwarded to Controller



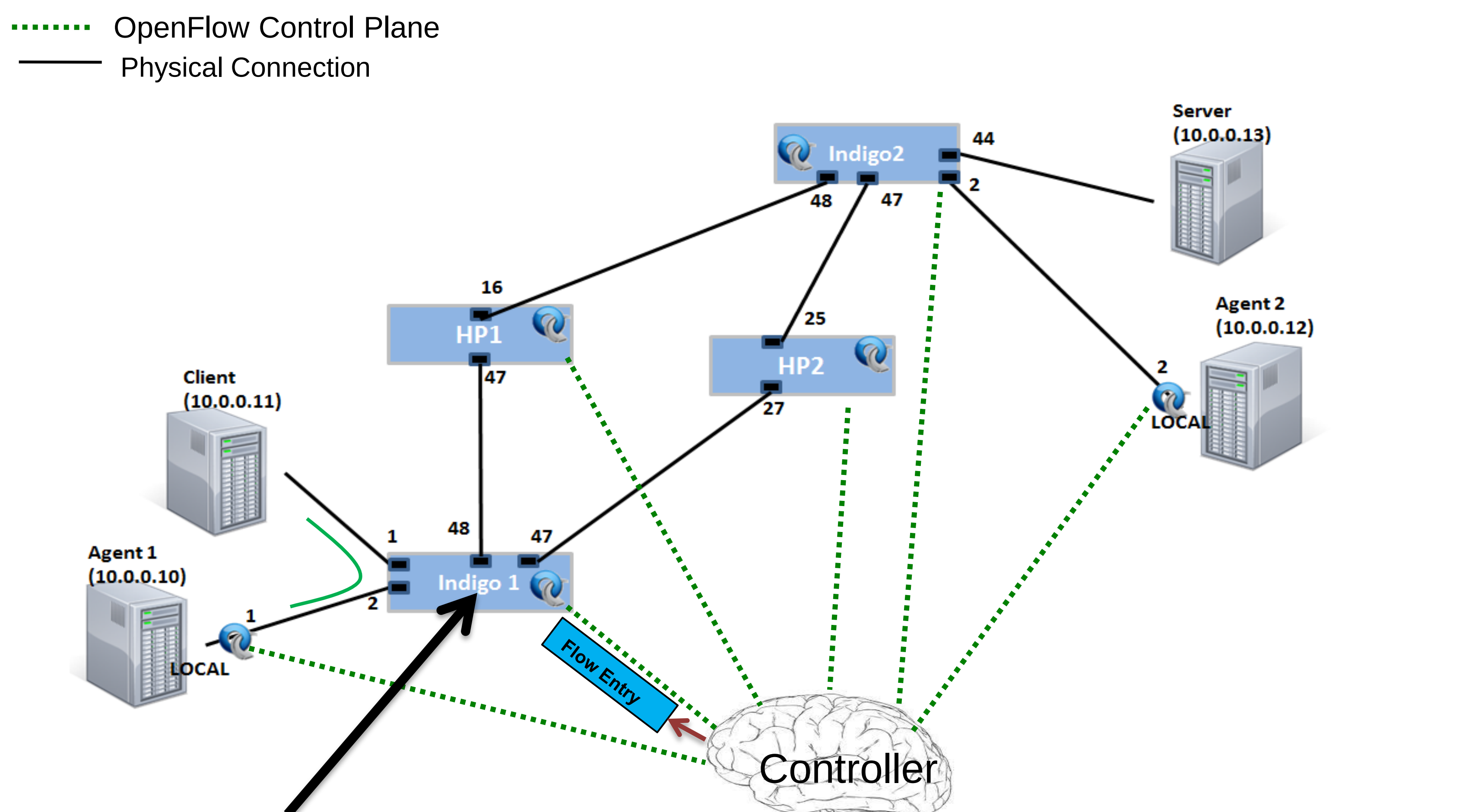
Controller Handles Packet



Controller Informs Agents



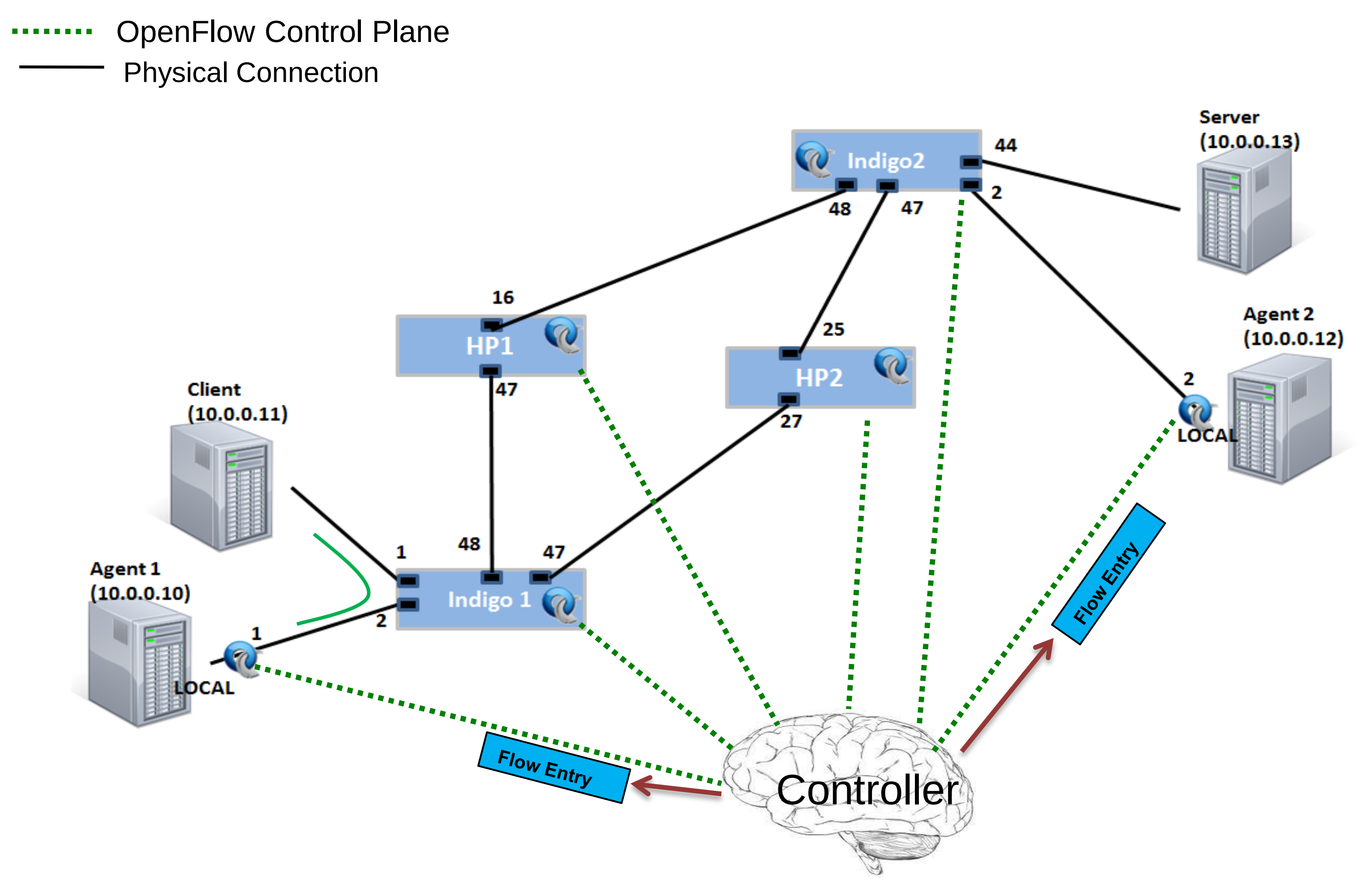
Redirecting traffic from Client to Agent 1



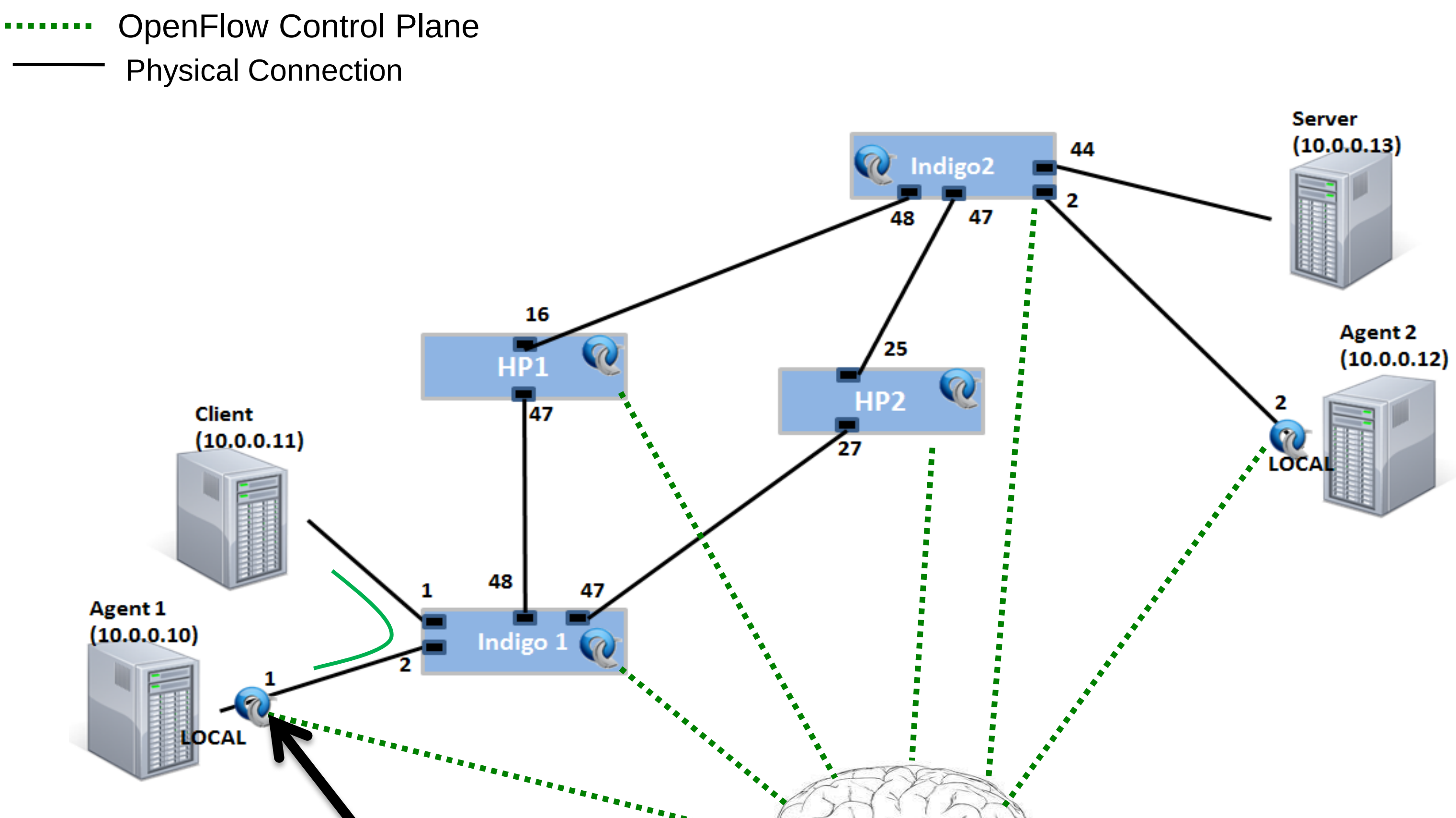
in_port=1,dl_vlan=0xffff,dl_vlan_pcp=0x00,dl_src=00:1f:29:32:91:99,dl_dst=00:1b:21:6a:85:88,nw_src=10.0.0.11,nw_dst=10.0.0.13,
nw_tos=0x00,tp_src=41922,tp_dst=5003,actions=output:2

in_port=2,dl_dst=00:1f:29:32:91:99,nw_src=10.0.0.13,nw_dst=10.0.0.11,tp_src=5003,tp_dst=41922,actions=output:1

More Flow Entries are installed



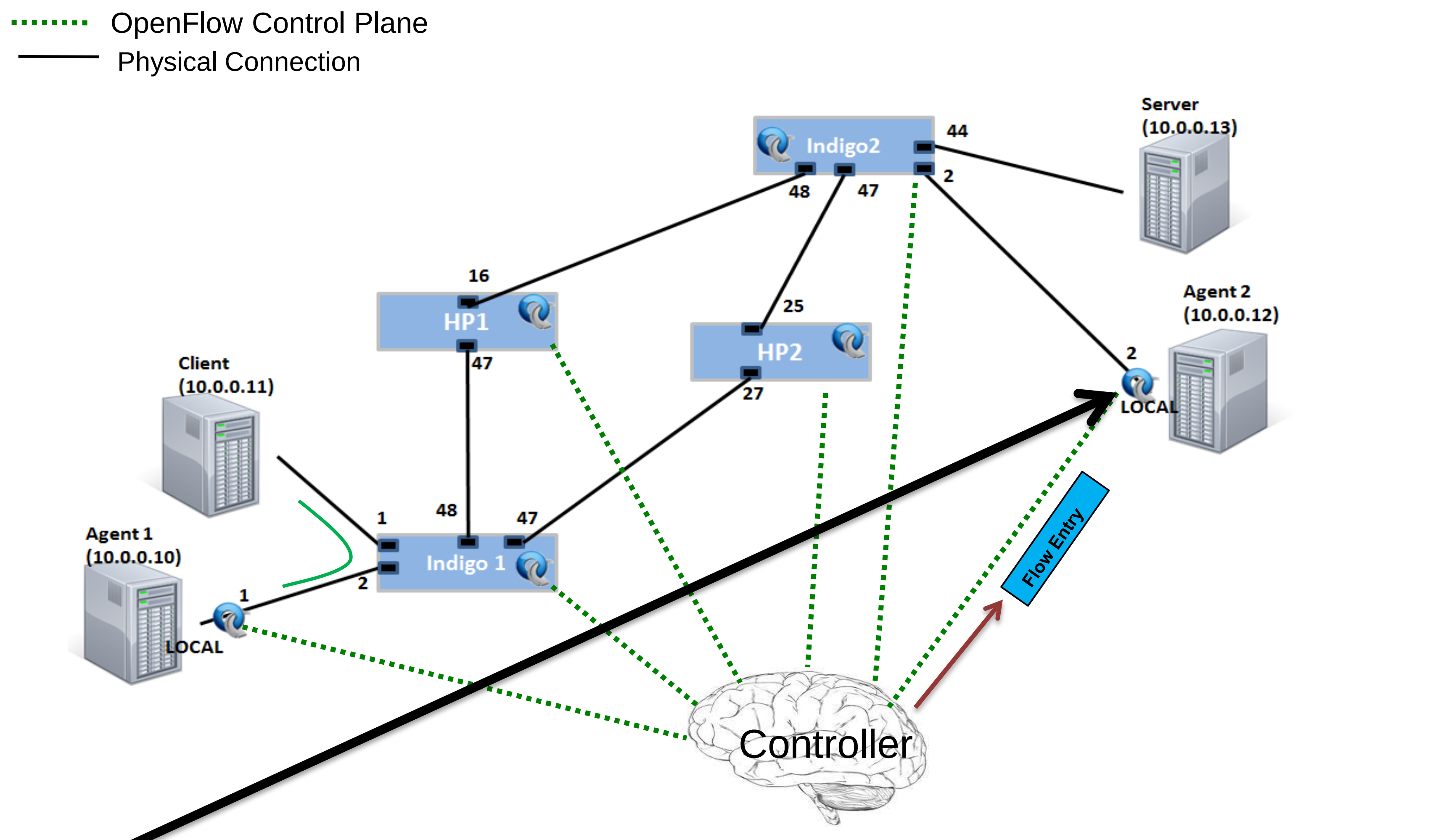
Flow Entry Modifies Packet Headers so Agent 1 can Accept Stream



in_port=1,vlan_tci=0x0000,dl_src=00:1f:29:32:91:99,dl_dst=00:1b:21:6a:85:88,nw_src=10.0.0.11,nw_dst=10.0.0.13,nw_tos=0,tp_src=41922,tp_dst=5003 actions=mod_dl_dst:00:1f:29:32:92:4d,mod_nw_dst:10.0.0.10,mod_tp_dst:9877,LOCAL

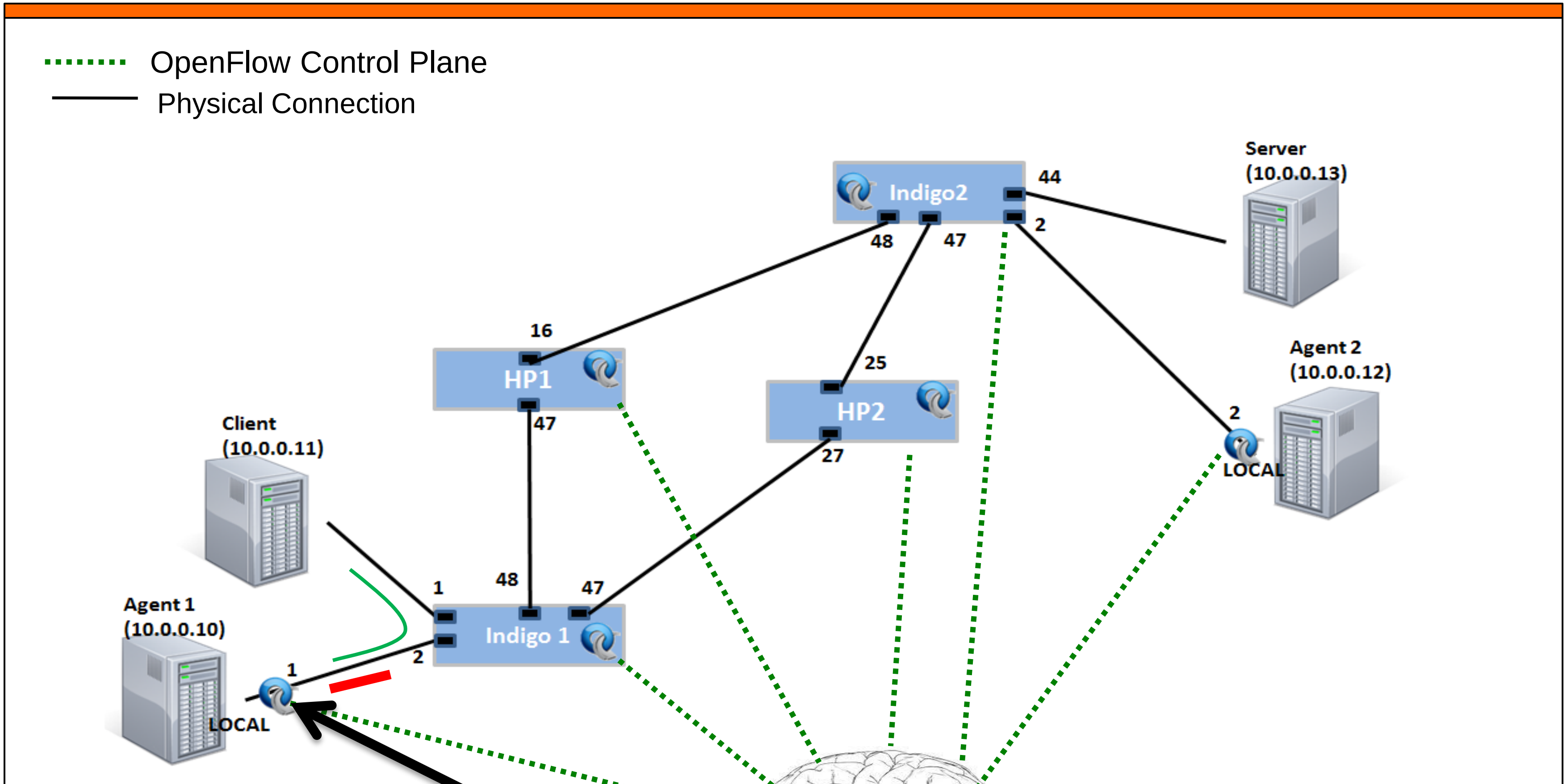
in_port=65534,dl_src=00:1f:29:32:92:4d,nw_src=10.0.0.10,nw_dst=10.0.0.11,tp_src=9877,tp_dst=41922 actions=mod_dl_src:00:1b:21:6a:85:88,mod_nw_src:10.0.0.13,mod_tp_src:5003,output:1

Flow Entry Modifies Packet Headers for Server to Agent 2



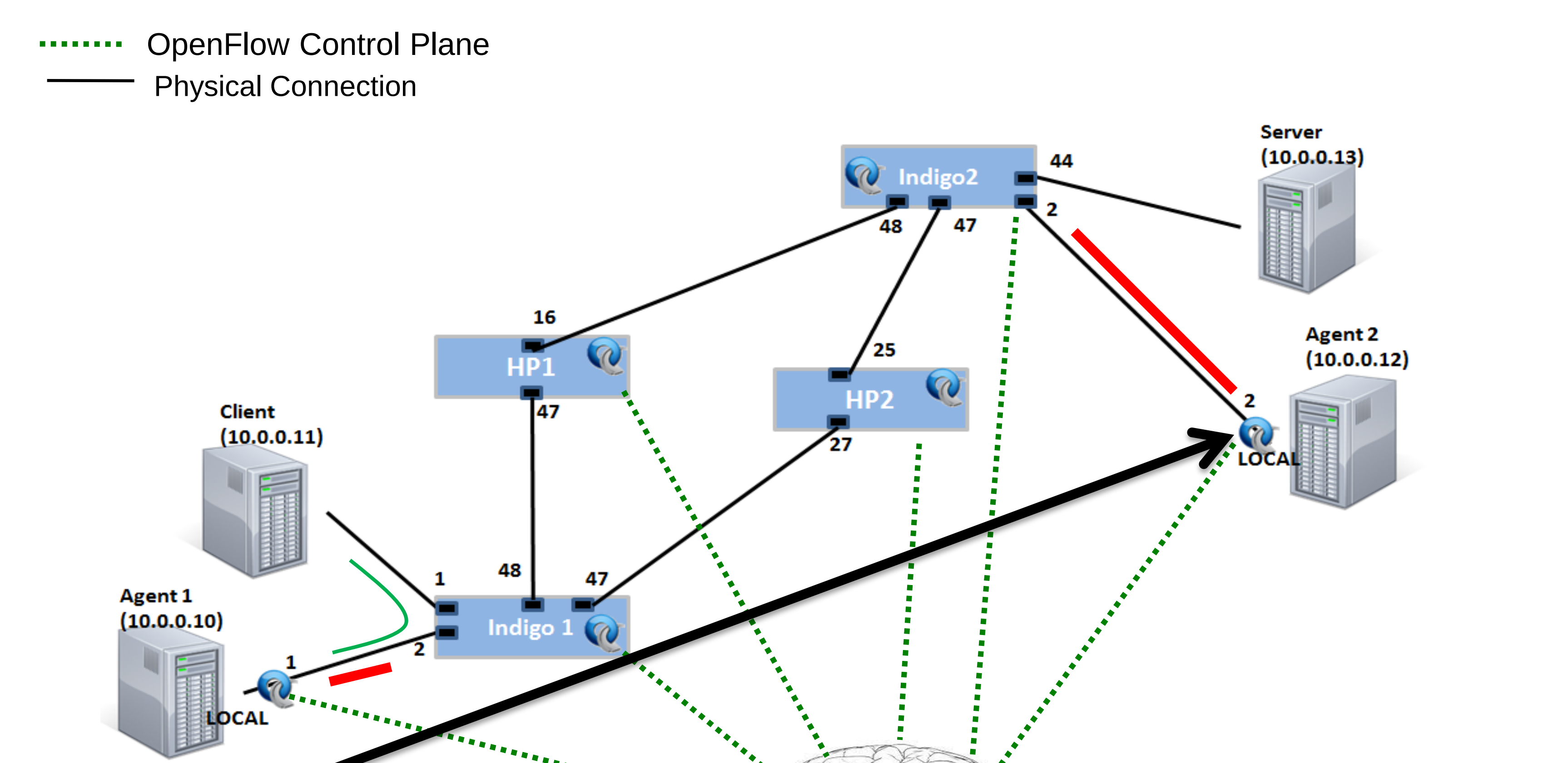
in_port=2,nw_src=10.0.0.13,nw_dst=10.0.0.11,tp_src=5003 actions=mod_dl_dst:00:1b:21:6b:50:df,mod_nw_dst:10.0.0.12,LOCAL

Flow Entries on Agent 1 for Agent to Agent Communication



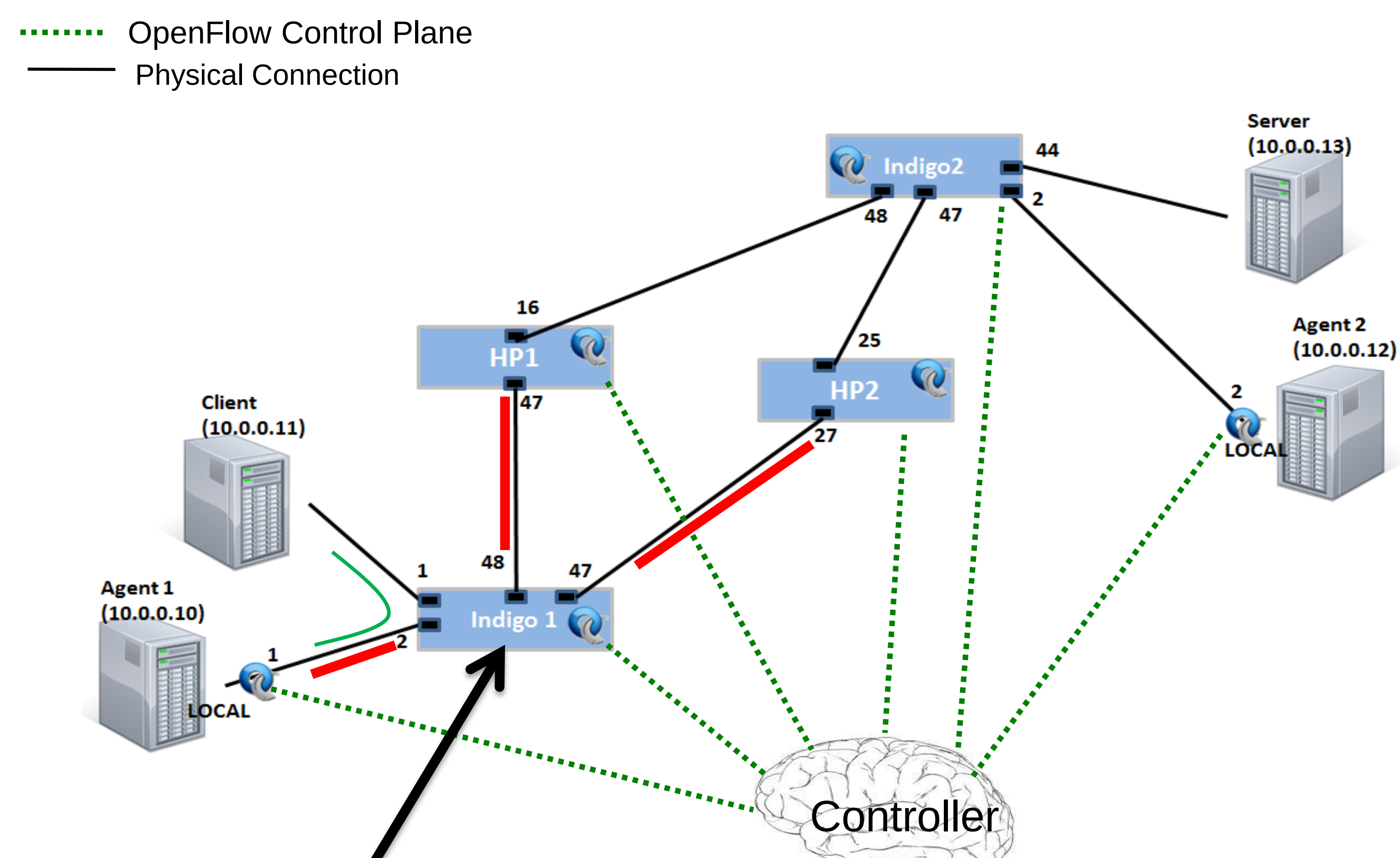
Agent1 to Agent 2 flow entries using 4 streams and 2 different paths (see different mod_dl_src/dst)
in_port=65534,nw_src=10.0.0.10,nw_dst=10.0.0.12,tp_dst=9881actions=mod_dl_src:00:1f:29:32:92:4f,mod_dl_dst:00:1b:21:6b:50:e1,output:1
in_port=65534,nw_src=10.0.0.10,nw_dst=10.0.0.12,tp_dst=9880actions=mod_dl_src:00:1f:29:32:92:4e,mod_dl_dst:00:1b:21:6b:50:e0,output:1
in_port=65534,nw_src=10.0.0.10,nw_dst=10.0.0.12,tp_dst=9879actions=mod_dl_src:00:1f:29:32:92:4f,mod_dl_dst:00:1b:21:6b:50:e1,output:1
in_port=65534,nw_src=10.0.0.10,nw_dst=10.0.0.12,tp_dst=9878actions=mod_dl_src:00:1f:29:32:92:4e,mod_dl_dst:00:1b:21:6b:50:e0,output:1
Agent 2 to Agent 1 flow entry resetting the MAC addresses to be the correct values
in_port=1,nw_src=10.0.0.12,nw_dst=10.0.0.10 actions=mod_dl_src:00:1b:21:6b:50:df,mod_dl_dst:00:1f:29:32:92:4d,LOCAL

Flow Entries on Agent 2 for Agent to Agent Communication



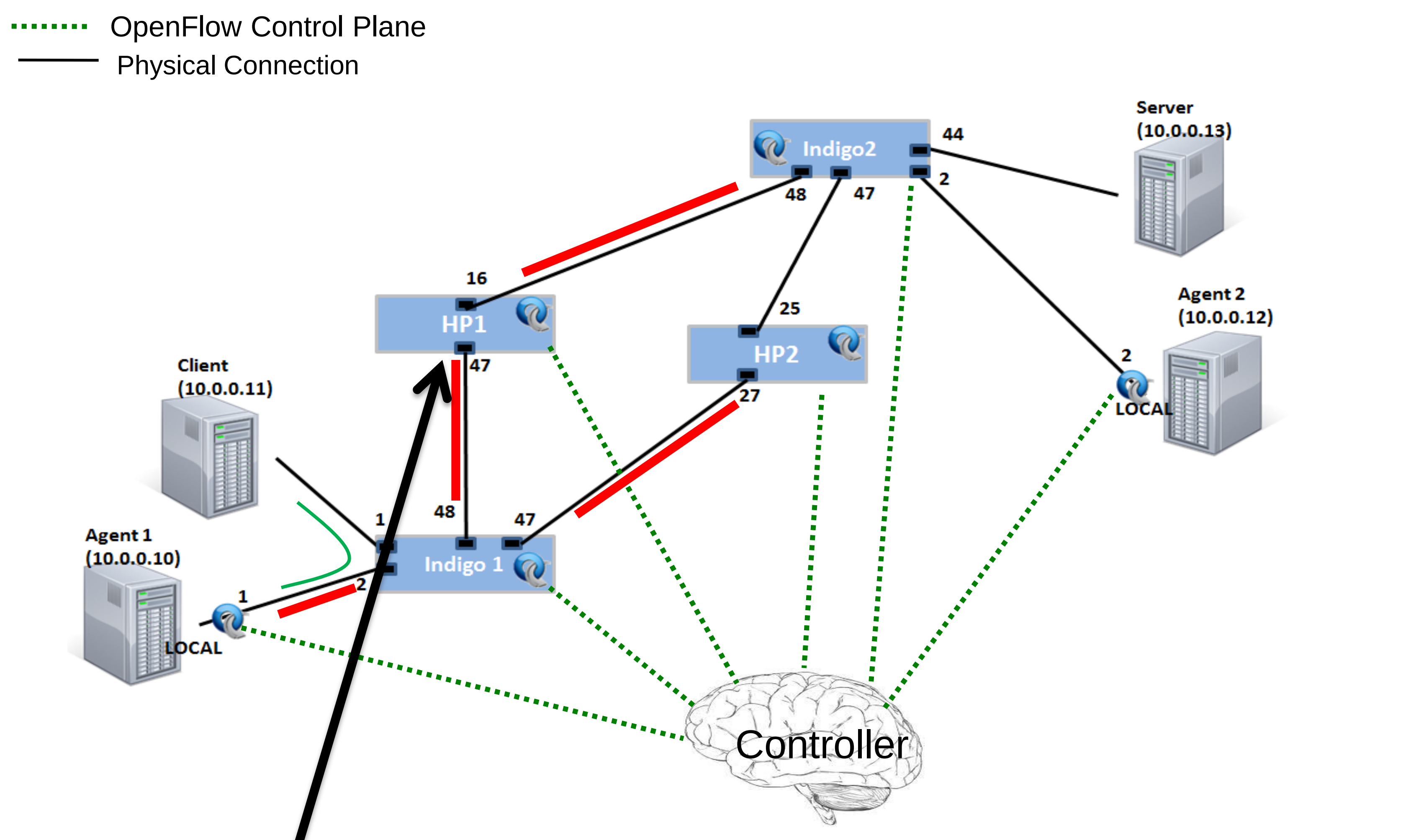
Agent 2 to Agent 1 flow entries using 4 streams and 2 different paths (see different mod_dl_dst)
in_port=65534,nw_src=10.0.0.12,nw_dst=10.0.0.10,tp_src=9880actions=*mod_dl_src:00:1b:21:6b:50:e0,mod_dl_dst:00:1f:29:32:92:4e*,output:2 in_port=65534,nw_src=10.0.0.12,nw_dst=10.0.0.10,tp_src=9878actions=*mod_dl_src:00:1b:21:6b:50:e0,mod_dl_dst:00:1f:29:32:92:4e*,output:2 in_port=65534,nw_src=10.0.0.12,nw_dst=10.0.0.10,tp_src=9881actions=*mod_dl_src:00:1b:21:6b:50:e1,mod_dl_dst:00:1f:29:32:92:4f*,output:2 in_port=65534,nw_src=10.0.0.12,nw_dst=10.0.0.10,tp_src=9879actions=mod_dl_src:00:1b:21:6b:50:e1,mod_dl_dst:00:1f:29:32:92:4f,output:2
Agent 1 to Agent 2 flow entry resetting the MAC addresses to be the correct values
in_port=2,nw_src=10.0.0.10,nw_dst=10.0.0.12 actions=mod_dl_src:00:1f:29:32:92:4d,mod_dl_dst:00:1b:21:6b:50:df, LOCAL

Flow Entries on Core for Agent to Agent Communication



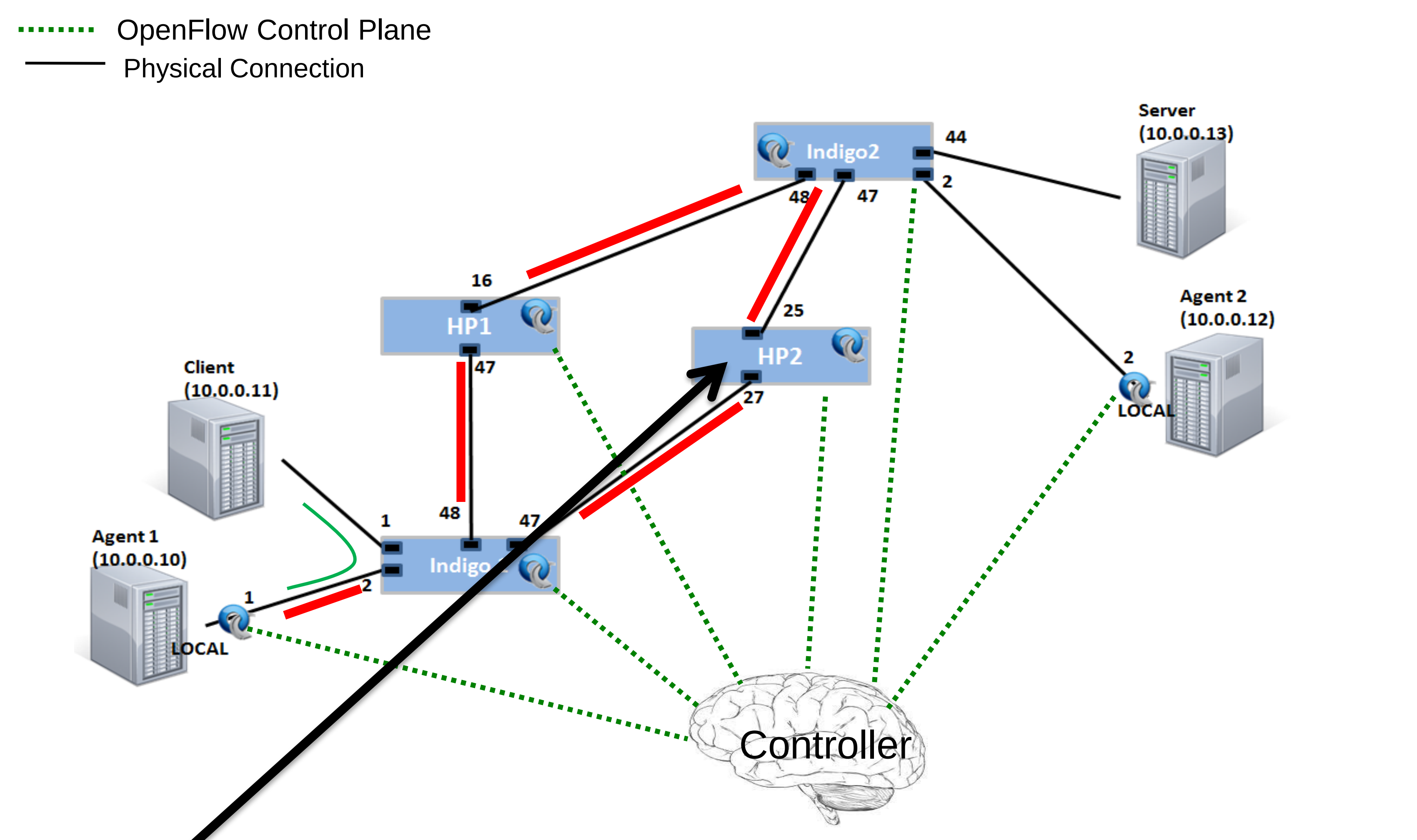
in_port=2,dl_src=00:1f:29:32:92:4f,dl_dst=00:1b:21:6b:50:e1nw_src=10.0.0.10,nw_dst=10.0.0.12,actions=output:48
in_port=2,dl_src=00:1f:29:32:92:4e,dl_dst=00:1b:21:6b:50:e0,nw_src=10.0.0.10,nw_dst=10.0.0.12,actions=output:47
in_port=48,nw_src=10.0.0.12,nw_dst=10.0.0.10,actions=output:2
in_port=47,nw_src=10.0.0.12,nw_dst=10.0.0.10,actions=output:2

Flow Entries on Core for Agent to Agent Communication



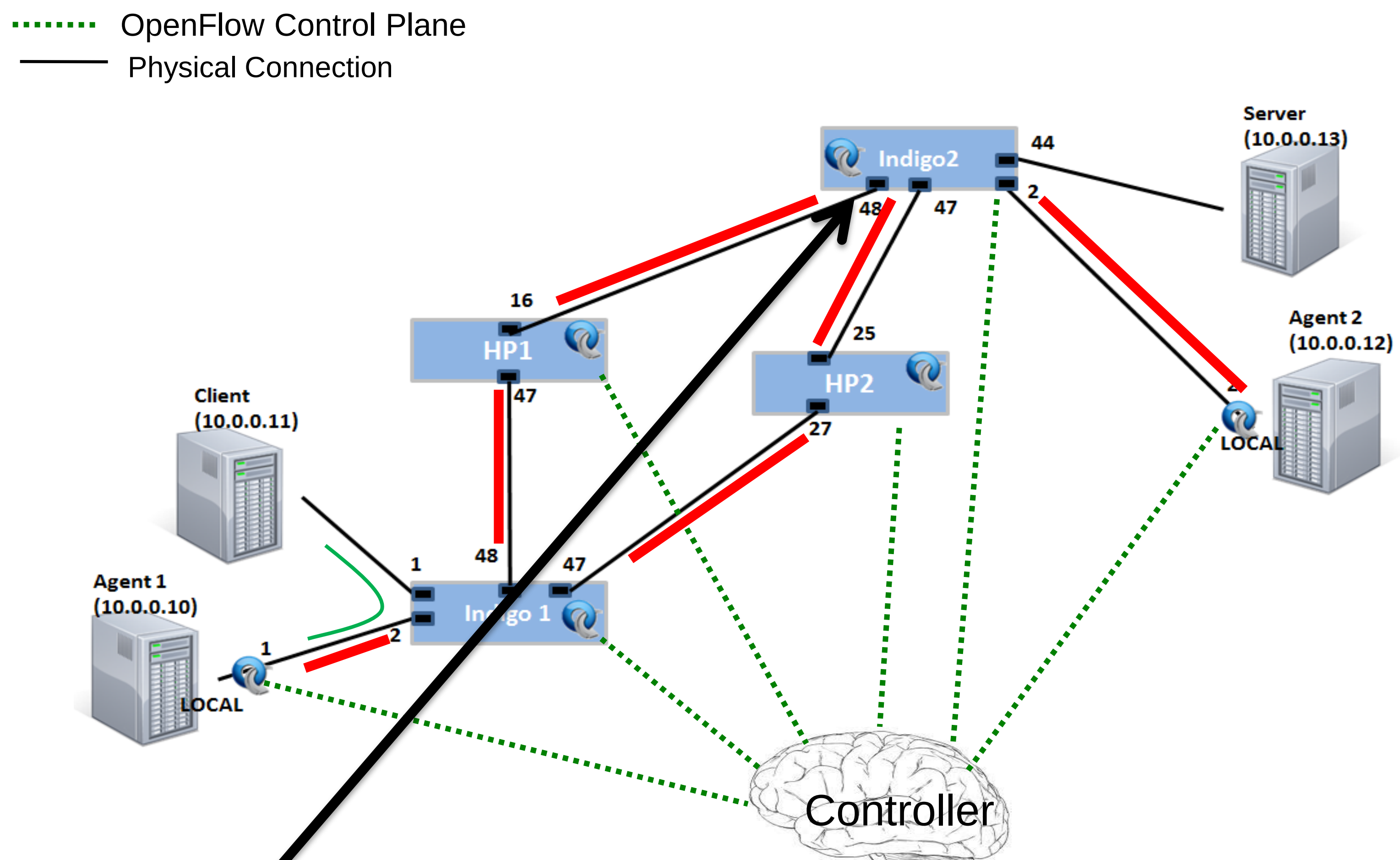
in_port=16,dl_src=00:1b:21:6b:50:e1,dl_dst=00:1f:29:32:92:4f,nw_src=10.0.0.12,nw_dst=10.0.0.10,actions=output:47
in_port=47,dl_src=00:1f:29:32:92:4f,dl_dst=00:1b:21:6b:50:e1,nw_src=10.0.0.10,nw_dst=10.0.0.12,actions=output:16

Flow Entries on Core for Agent to Agent Communication



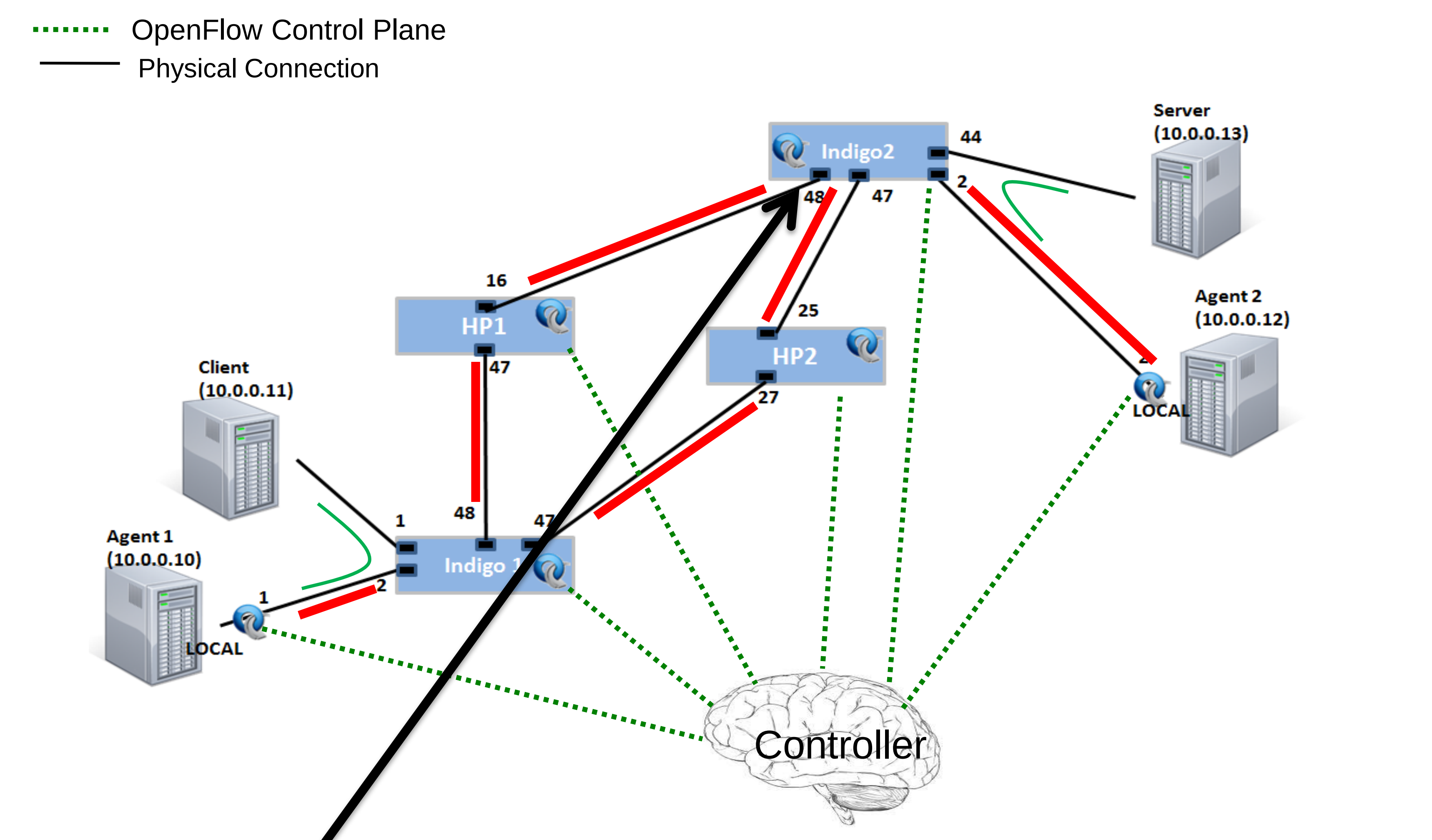
in_port=25,dl_src=00:1b:21:6b:50:e0,dl_dst=00:1f:29:32:92:4e,nw_src=10.0.0.12,nw_dst=10.0.0.10,actions=output:27
in_port=27,dl_src=00:1f:29:32:92:4e,dl_dst=00:1b:21:6b:50:e0,nw_src=10.0.0.10,nw_dst=10.0.0.12,actions=output:25

Flow Entries on Core for Agent to Agent Communication



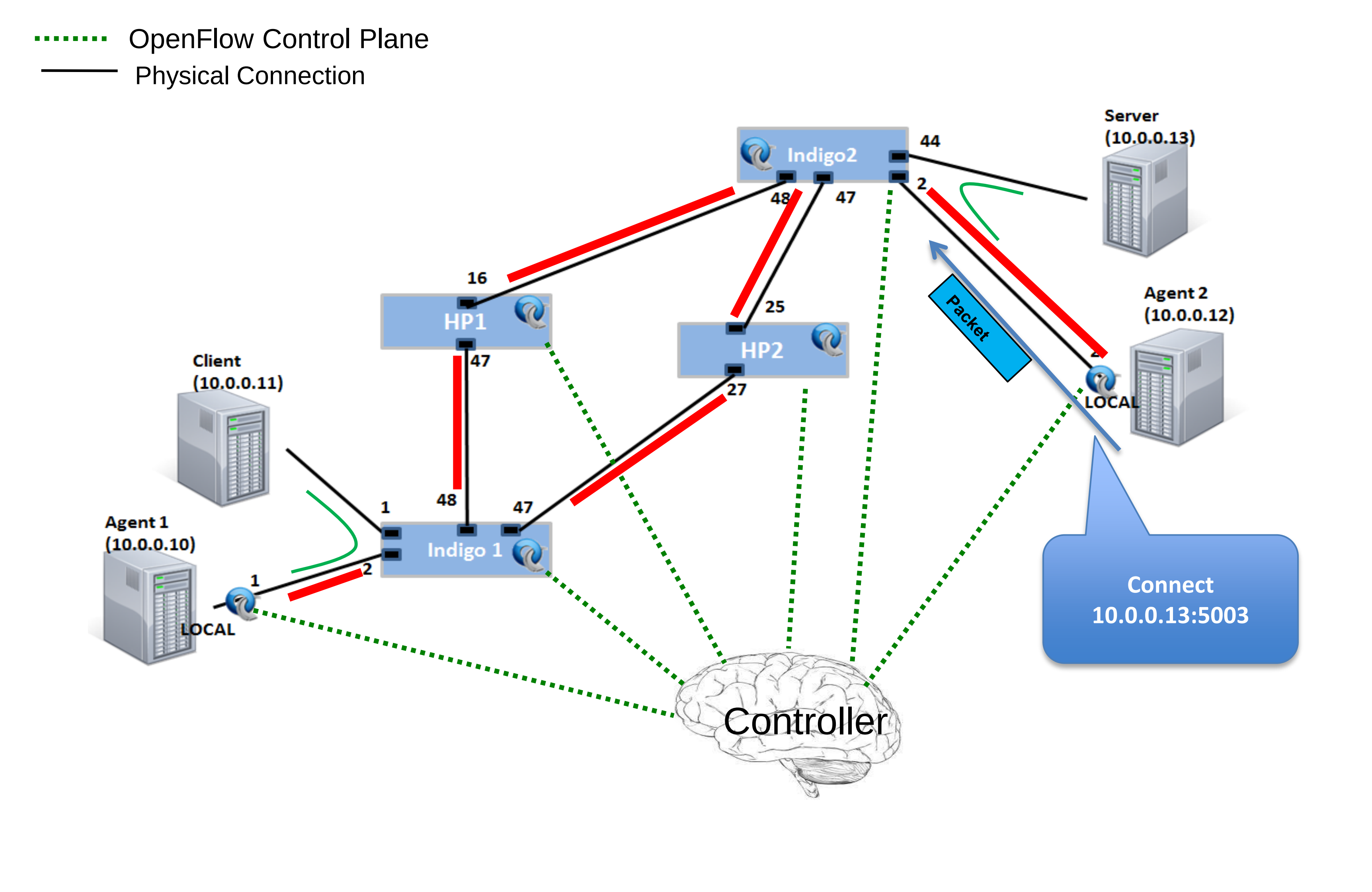
in_port=2,dl_src=00:1b:21:6b:50:e1,dl_dst=00:1f:29:32:92:4f,nw_src=10.0.0.12,nw_dst=10.0.0.10,actions=output:48
in_port=2,dl_src=00:1b:21:6b:50:e0,dl_dst=00:1f:29:32:92:4e,nw_src=10.0.0.12,nw_dst=10.0.0.10,actions=output:47
in_port=48,nw_src=10.0.0.10,nw_dst=10.0.0.12,actions=output:2
in_port=47,nw_src=10.0.0.10,nw_dst=10.0.0.12,actions=output:2

Flow Entries to Forward Between Server and Agent 2

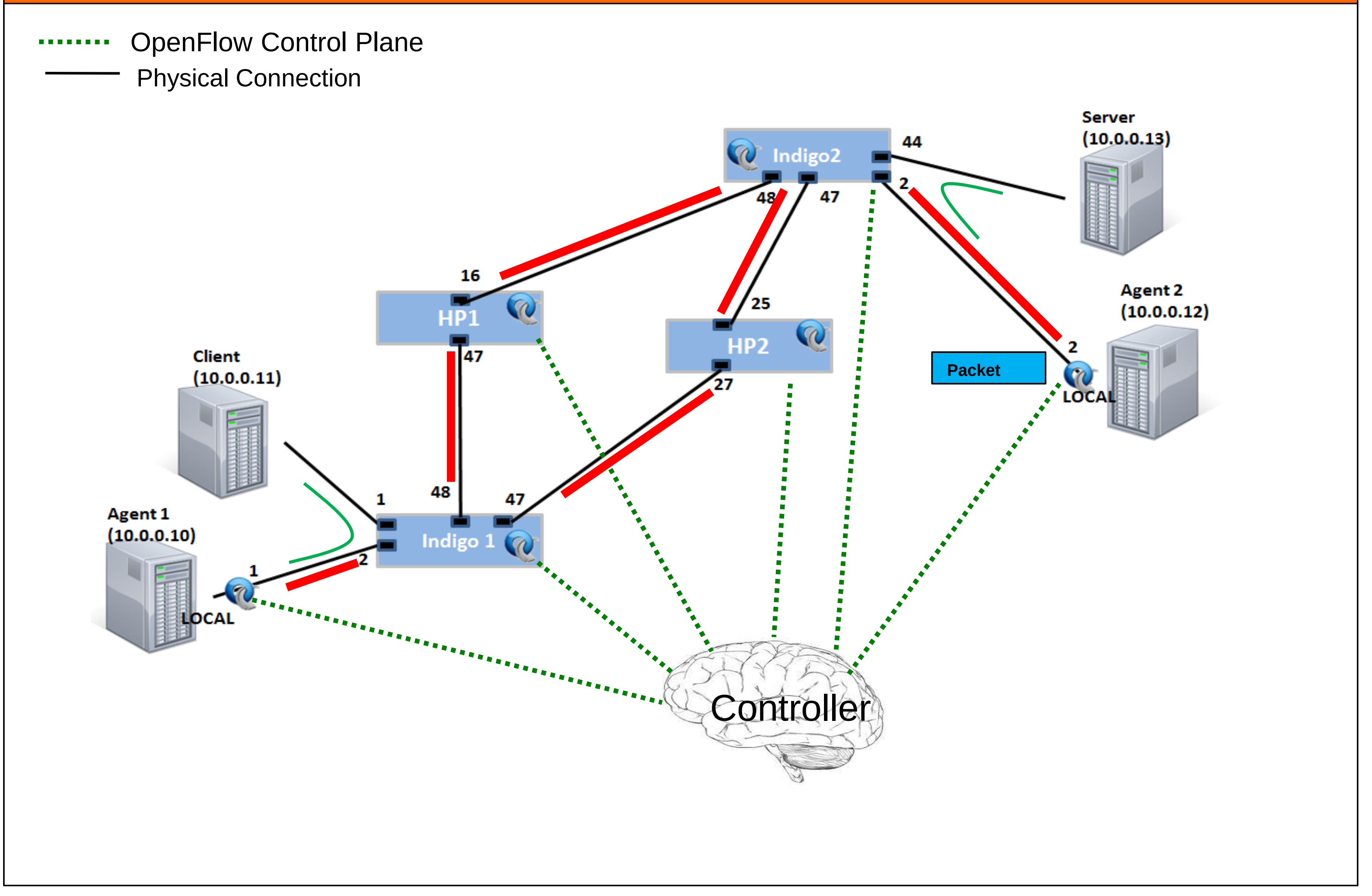


in_port=2,nw_src=10.0.0.11,nw_dst=10.0.0.13,tp_dst=5003,actions=output:44
in_port=44,nw_src=10.0.0.13,nw_dst=10.0.0.11,tp_src=5003,actions=output:2

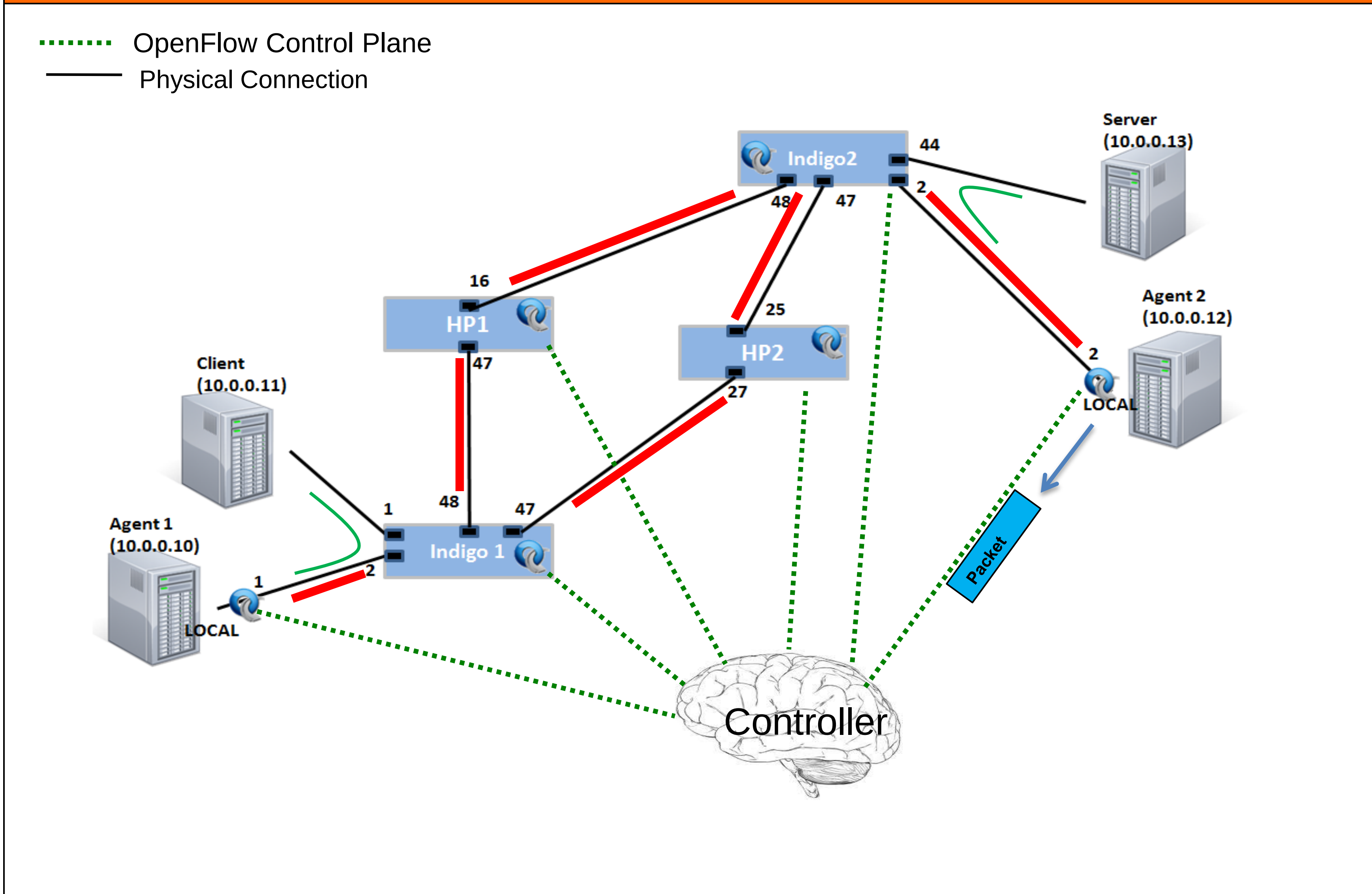
Agent 2 Initiates TCP connection to Server



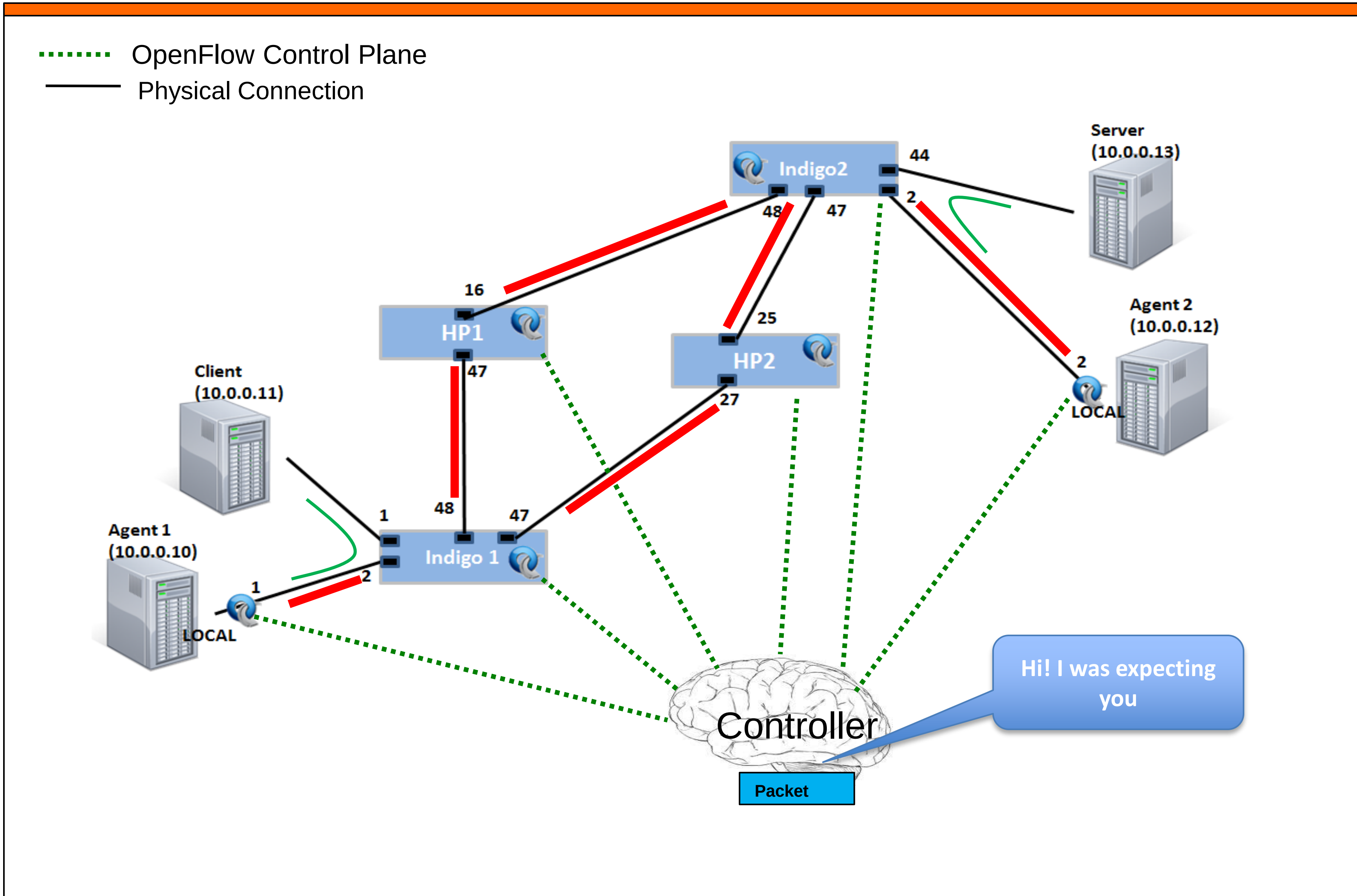
Packet reaches OpenFlow Datapath



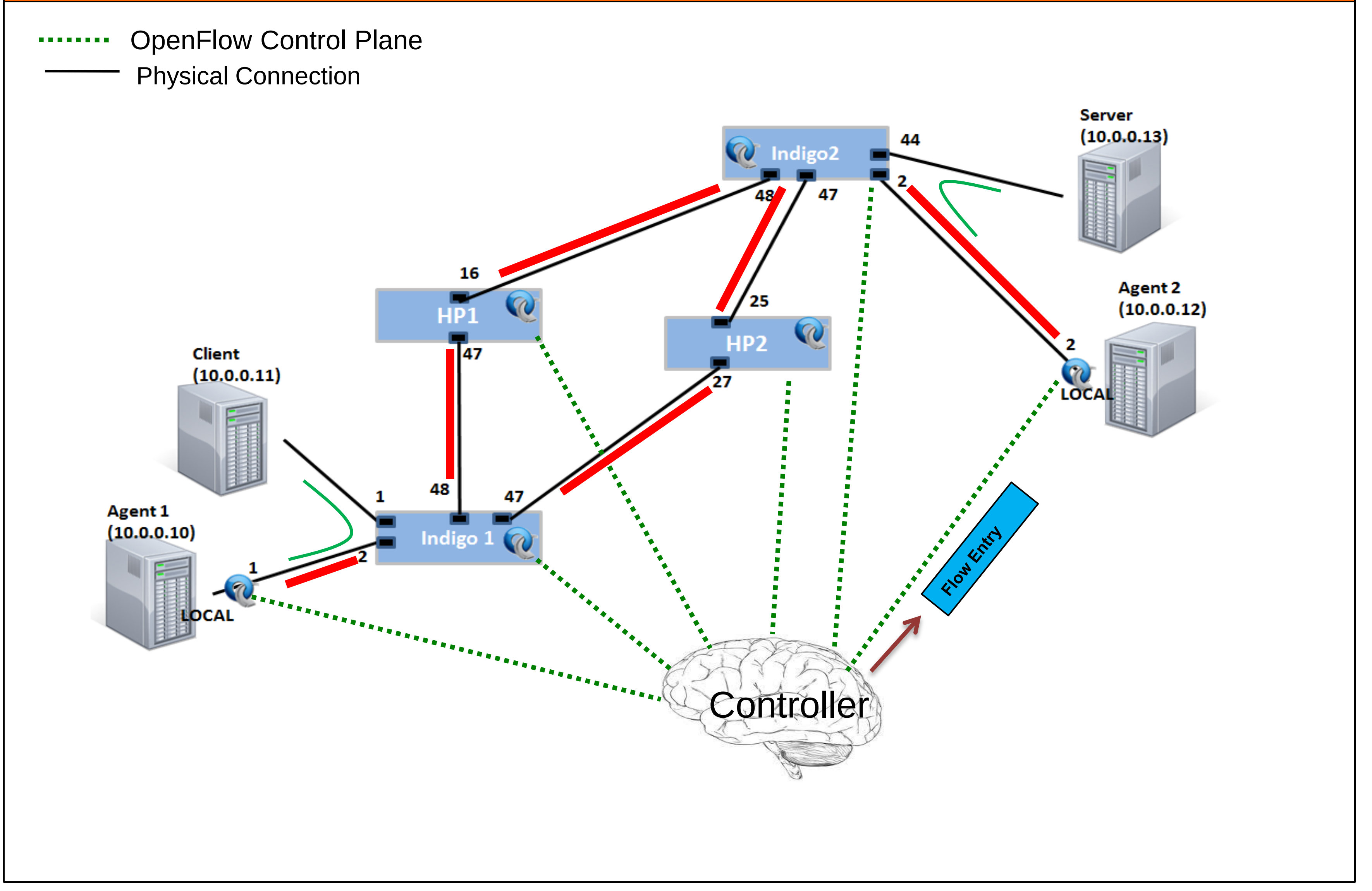
Forwarded to controller



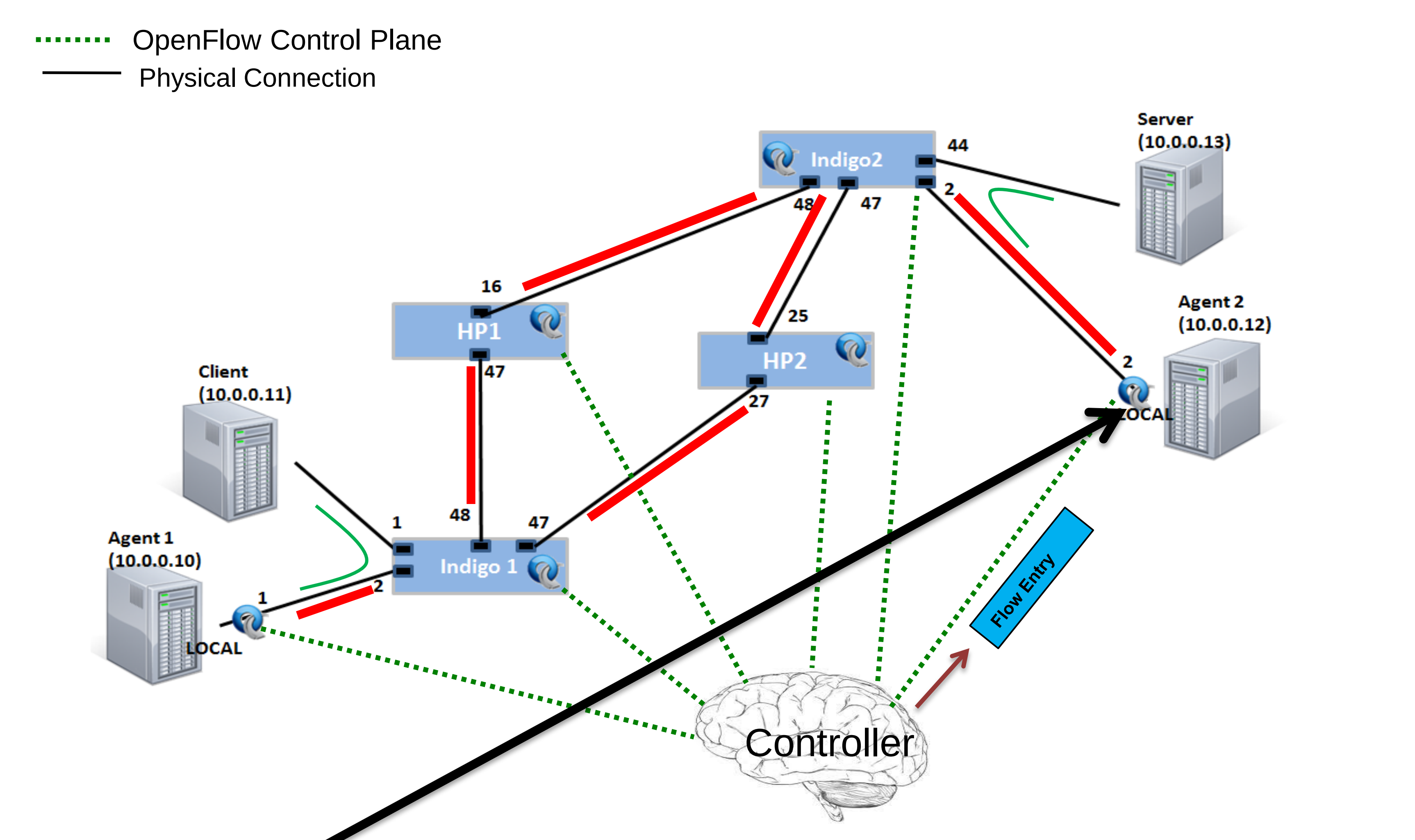
Controller Handles packet



Installs Flow Entry

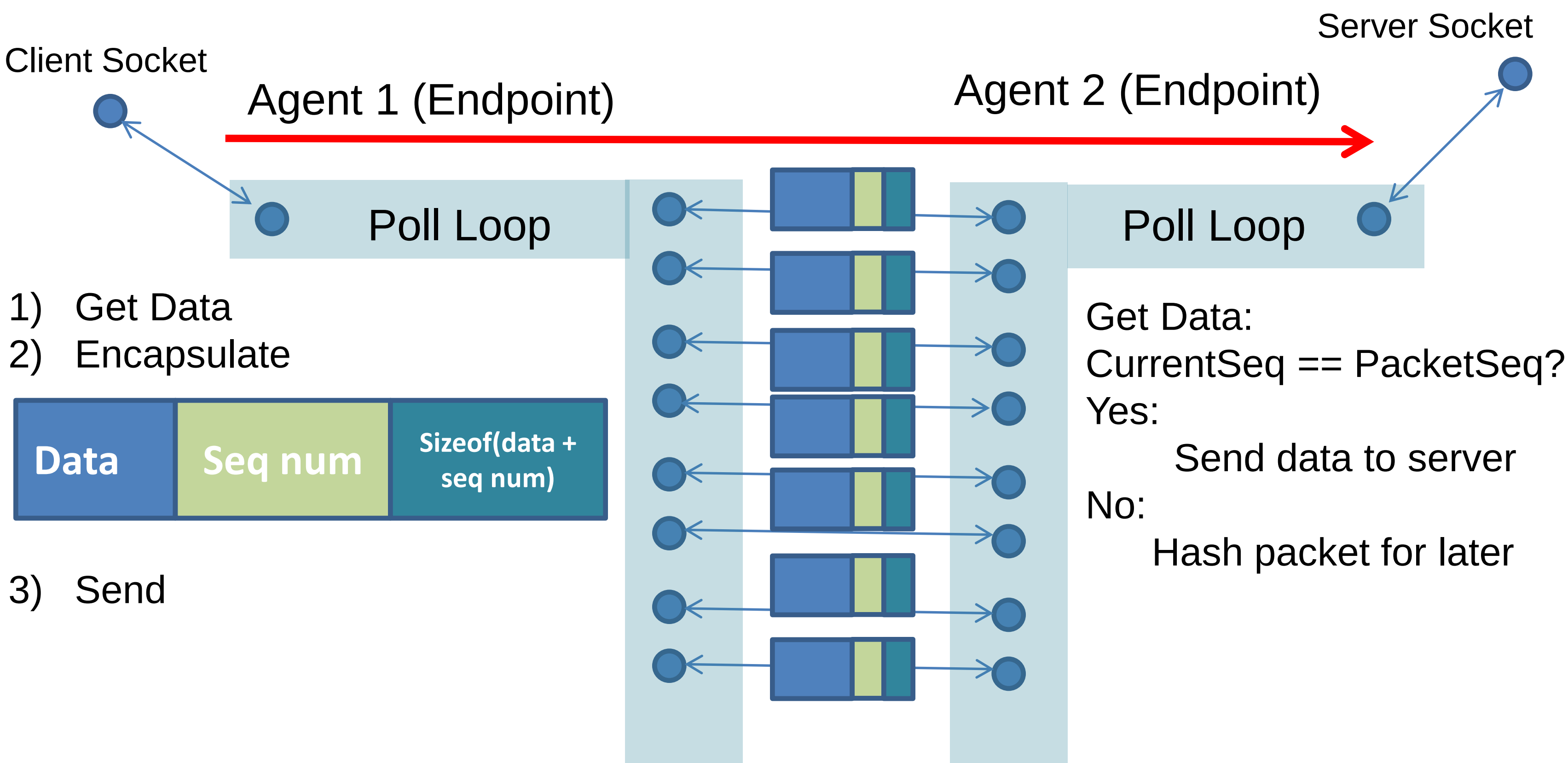


Last Flow Entry for Agent 2 to Server



in_port=65534,vlan_tci=0x0000,dl_src=00:1b:21:6b:50:df,dl_dst=00:1b:21:6a:85:88,nw_src=10.0.0.12,nw_dst=10.0.0.13,nw_tos=0,
tp_src=47489,tp_dst=5003 actions=mod_dl_src:00:1f:29:32:91:99,mod_nw_src:10.0.0.11,output:2

SOS Sockets


Client Socket
Agent 1 (Endpoint)
Agent 2 (Endpoint)
Server Socket
Poll Loop
Poll Loop
1) Get Data
2) Encapsulate
Data
Seq num
Sizeof(data + seq num)
3) Send
Get Data:
CurrentSeq == PacketSeq?
Yes:
Send data to server
No:
Hash packet for later

SOS Transmission Protocol

- SOS uses Parallel TCP in order to relay data between agents.
- Exploits buffer limits by using multiple sockets.
- Sending window grows much faster using multiple sockets.
- Improves throughput in scenarios:
  - Lossy network
  - High Bandwidth Delay Product
  - Multiple paths present

GENI

- Large multipath test bed across the US using Internet2 and National LamdbaRail.
- Equipped with OpenFlow at the edge and in the core of the network.
- Allows full network visibility and control in core network.
- Series of different compute resources (dedicated, shared).
- Allows experimenters to test ideas on a real network
- Uses network slicing to allow multiple users

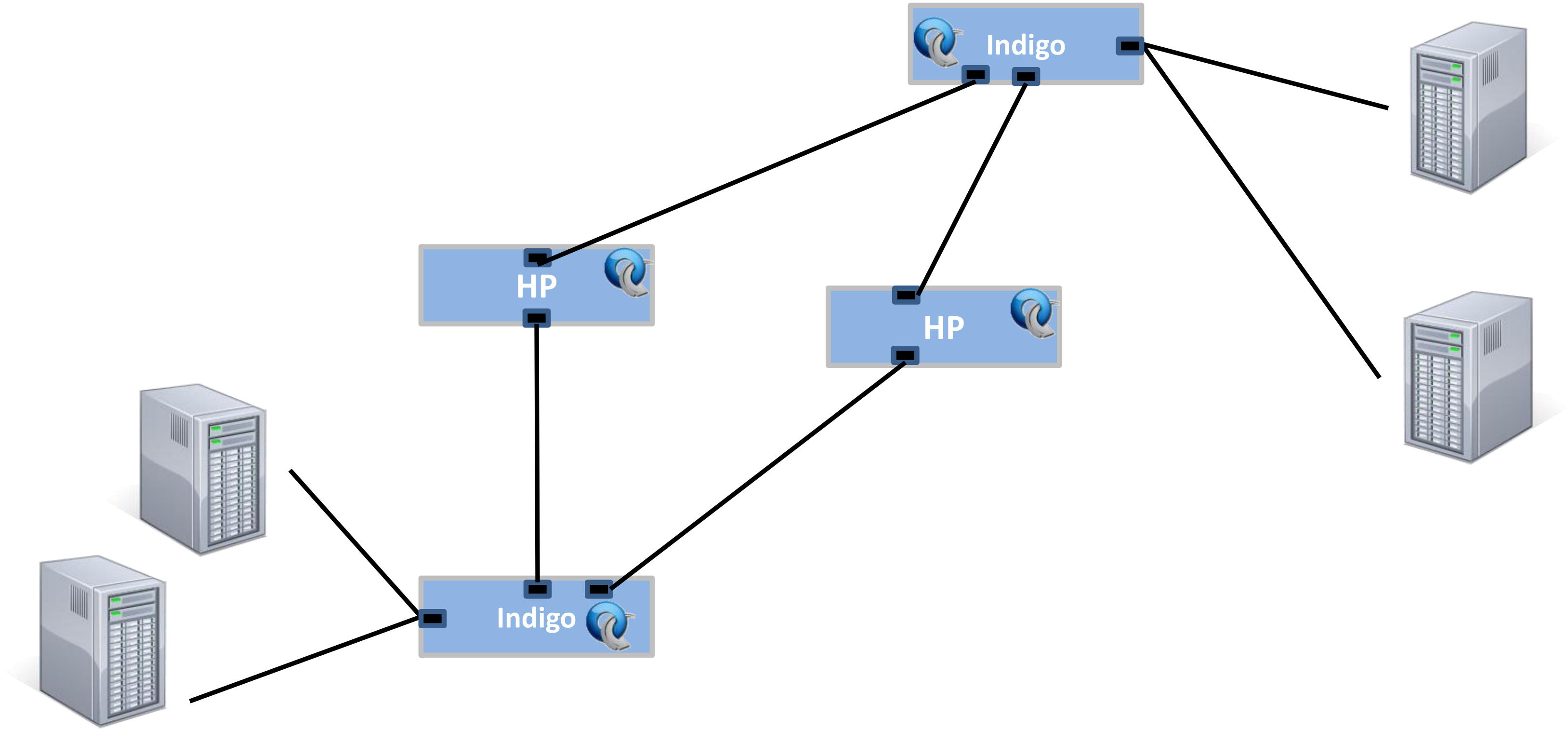
GENI Results

Configuration	Latency (ms)	TCP (Mbps)	UDP (Mbps)	SOS (Mbps)
Path 1 (short)	54	295	952	869
Path 2 (long)	160	95	952	788
Multipath				703

*Max average SOS throughput achieved when running for 3 minutes with varied number of sockets. (Could be better tuned to improve results)

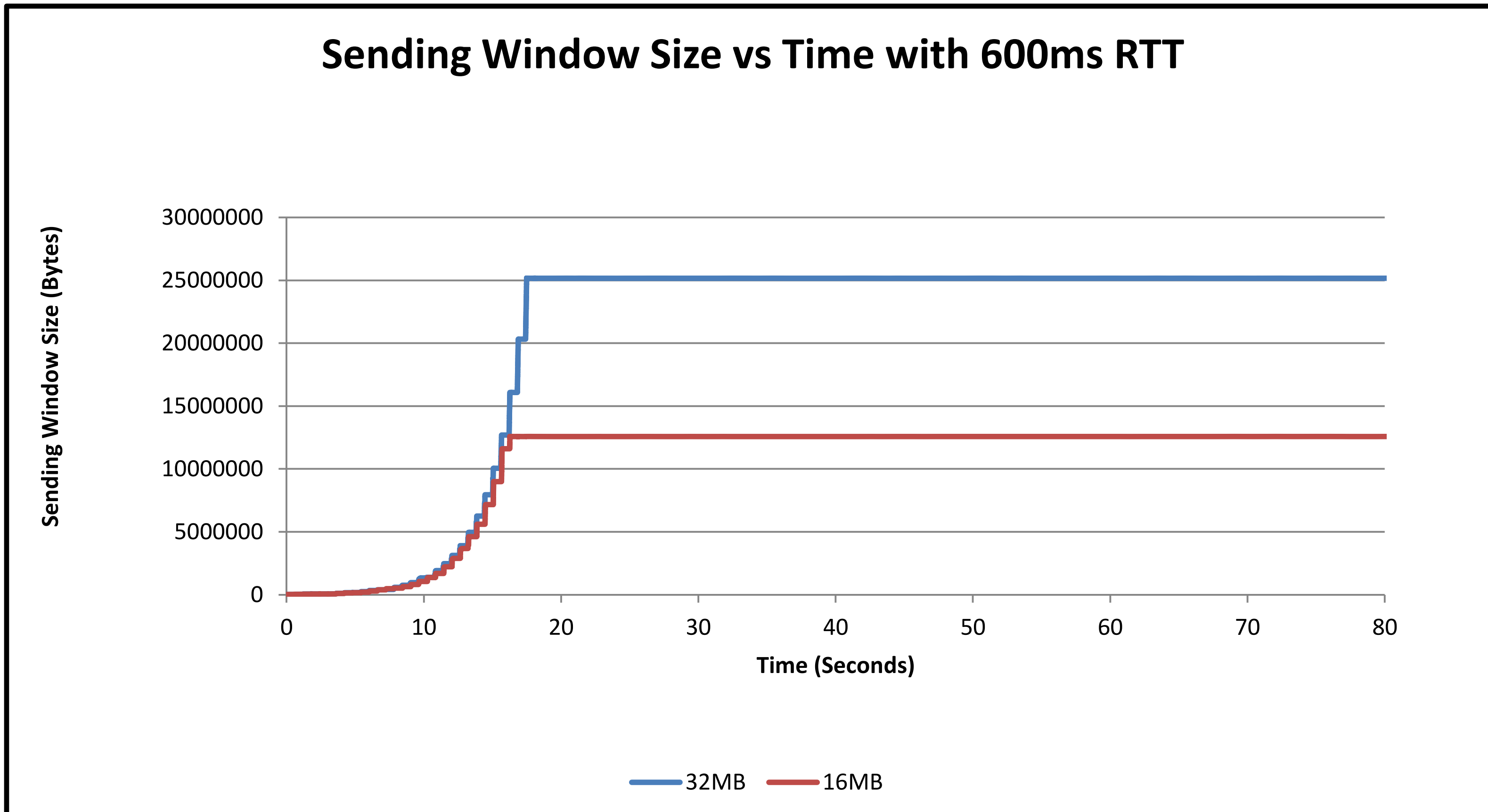
Test network

- In order to accurately measure network traffic a separate network was created and tc was used to emulate different network characteristics (latency, bandwidth, loss).



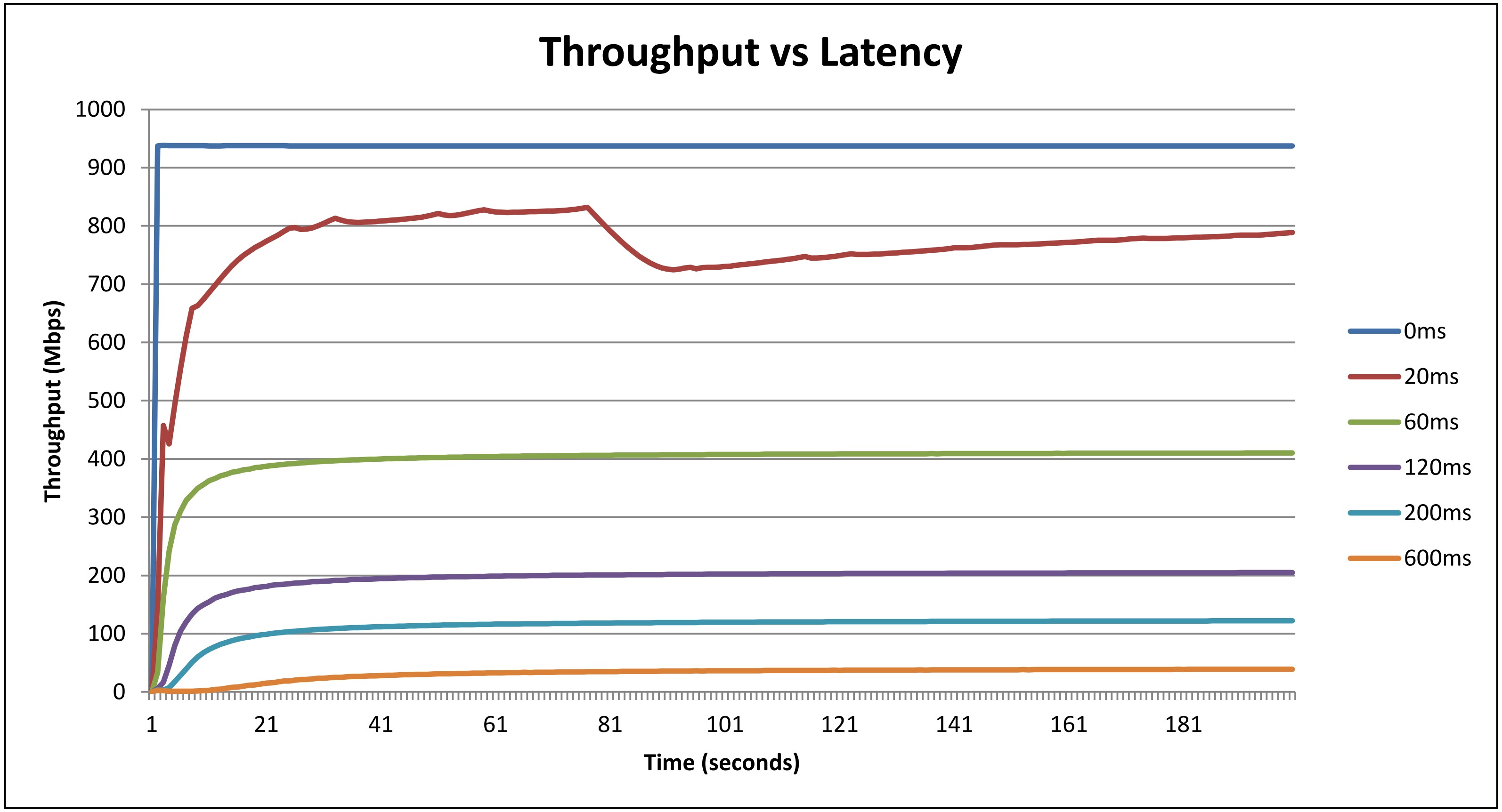
TCP Buffer Space

- Main limit to TCP throughput in large bandwidth delay product network is lack of buffer space. Test below shows that sending window grows exponentially until buffer limit is reached.



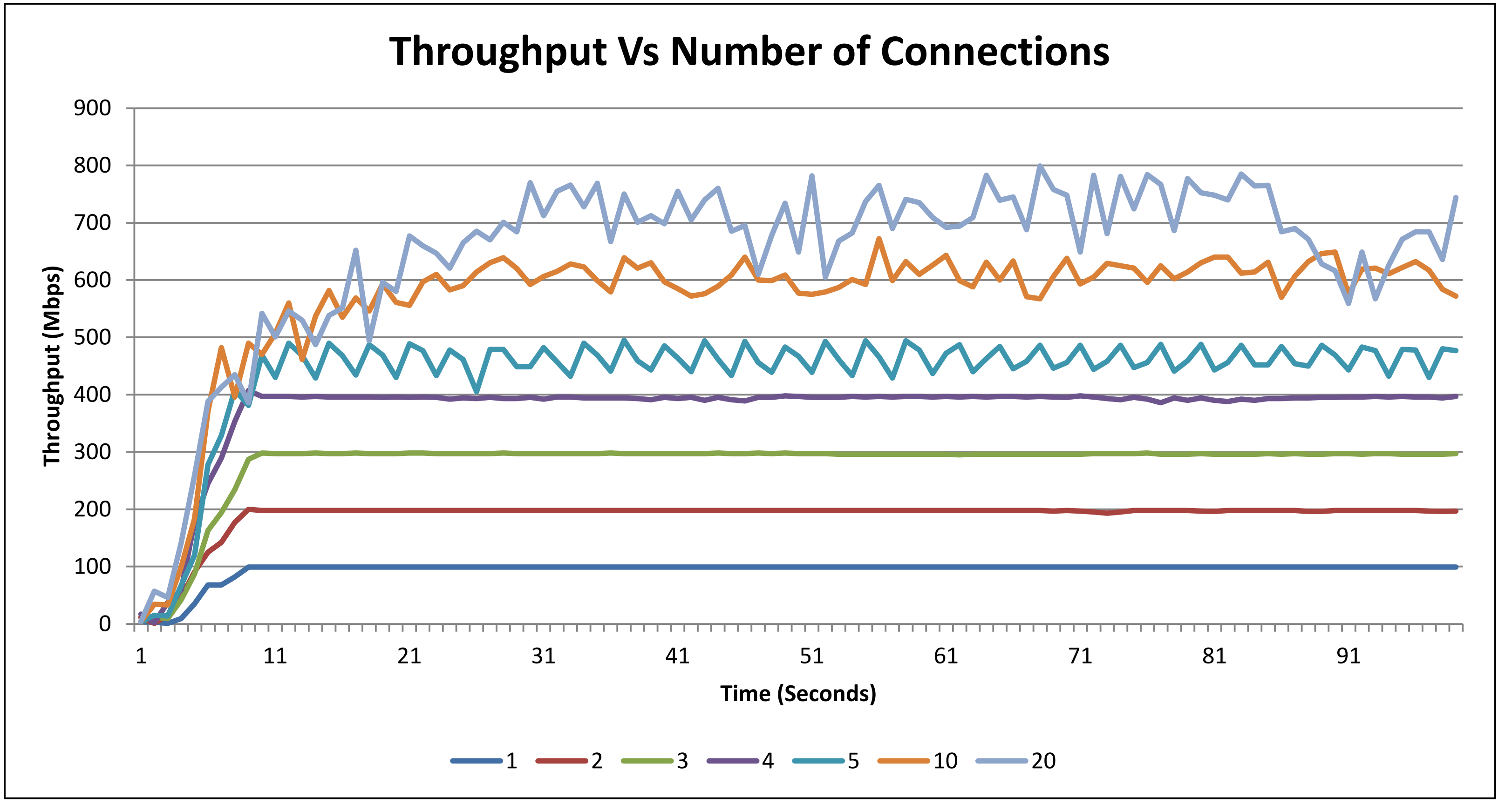
TCP Latency Throughput Growth Rate

- Latency affects the rate of which a TCP stream will grow since TCP window growth is updated based on RTT



Parallel TCP Solution

- Using multiple TCP sockets allows the buffer size limit to be exploited
- TCP slow-start is faster
- Graph shows linear growth until congestion occurs



Receiver Queue Benefits

- TCP suffers from head of line blocking
- Due to out of order arrivals at network layer
  - Retransmitted packets
  - Packet-level multipath routing
  - Route fluttering
  - And many other reasons
- Parallel TCP suffers from same issues in addition to at the application level
  - Out of order packet arrivals among TCP streams
- Receiver queue allows agent to receive data from streams while others are blocked
- Helps increase performance when:
  - Multiple paths of varied latency and bandwidth are used
  - Lossy network paths are present

Latency Effect on Throughput with Receiver Queue

- Network setup with two 300Mbit paths
- Latency was then varied on one path and kept constant on the other
- Queue size represents the number of application level packets that can be stored

Throughput (Mbps) vs Number of streams								
# Streams	1		2		4		6	
Queue Size	1	1000	1	1000	1	1000	1	1000
25-25 (ms)	291	291	580	580	580	584	580	585
50-25 (ms)	198	199	432	486	567	562	569	572
100-25 (ms)	97	96	304	380	452	478	524	541
200-25 (ms)	44	44	157	250	309	336	371	390
400-25 (ms)	19	19	60	122	139	209	183	278

*Note: when 1 stream is used it takes path of higher latency

Average Queue Length (Bytes) Comparing Paths of Varied Latency (2 Streams)									
25-25(ms)		50-25(ms)		100-25(ms)		200-25(ms)		400-25(ms)	
.00176	.00142	.00101	.04039	.00115	.32321	.01184	.623	.1411	.4805

Receiver Queue Provides Little Benefit using Paths of Same Latency

- Receiver queue helps increase performance when multiple paths of varied latency are used
- Need to have enough queue space to buffer the difference in bandwidth delay product of paths in order to avoid blocking

0% Performance Improvement

Throughput (Mbps) vs Number of streams								
# Streams	**1**		**2**		**4**		**6**	
Queue Size	**1**	**1000**	**1**	**1000**	**1**	**1000**	**1**	**1000**
25-25 (ms)	291	291	580	580	580	584	580	585
50-25 (ms)	198	199	432	486	567	562	569	572
100-25 (ms)	97	96	304	380	452	478	524	541
200-25 (ms)	44	44	157	250	309	336	371	390
400-25 (ms)	19	19	60	122	139	209	183	278

*Note: when 1 stream is used it takes path of higher latency

Average Queue Length (Bytes) Comparing Paths of Varied Latency (2 Streams)									
25-25(ms)		50-25(ms)		100-25(ms)		200-25(ms)		400-25(ms)	
.00176	.00142	.00101	.04039	.00115	.32321	.01184	.623	.1411	.4805

Queue Yields Performance Increase When Path Latency Varies

- Receiver queue helps increase performance when multiple paths of varied latency are used.
- Need to have enough queue space to buffer the difference in bandwidth delay product of paths in order to avoid blocking.

12.5% Performance Improvement

Throughput (Mbps) vs Number of streams								
# Streams	1		2		4		6	
Queue Size	1	1000	1	1000	1	1000	1	1000
25-25 (ms)	291	291	580	580	580	584	580	585
50-25 (ms)	198	199	432	486	567	562	569	572
100-25 (ms)	97	96	304	380	452	478	524	541
200-25 (ms)	44	44	157	250	309	336	371	390
400-25 (ms)	19	19	60	122	139	209	183	278

*Note: when 1 stream is used it takes path of higher latency

Average Queue Length (Bytes) Comparing Paths of Varied Latency (2 Streams)									
25-25(ms)		50-25(ms)		100-25(ms)		200-25(ms)		400-25(ms)	
.00176	.00142	.00101	.04039	.00115	.32321	.01184	.623	.1411	.4805

Queue Yields Performance Increase When Path Latency Varies

- Receiver queue helps increase performance when multiple paths of varied latency are used.
- Need to have enough queue space to buffer the difference in bandwidth delay product of paths in order to avoid blocking.

25% Performance Improvement

Throughput (Mbps) vs Number of streams								
# Streams	1		2		4		6	
Queue Size	1	1000	1	1000	1	1000	1	1000
25-25 (ms)	291	291	580	580	580	584	580	585
50-25 (ms)	198	199	432	486	567	562	569	572
100-25 (ms)	97	96	304	380	452	478	524	541
200-25 (ms)	44	44	157	250	309	336	371	390
400-25 (ms)	19	19	60	122	139	209	183	278

*Note: when 1 stream is used it takes path of higher latency

Average Queue Length (Bytes) Comparing Paths of Varied Latency (2 Streams)									
25-25(ms)		50-25(ms)		100-25(ms)		200-25(ms)		400-25(ms)	
.00176	.00142	.00101	.04039	.00115	.32321	.01184	.623	.1411	.4805

Queue Yields Performance Increase When Path Latency Varies

- Receiver queue helps increase performance when multiple paths of varied latency are used.
- Need to have enough queue space to buffer the difference in bandwidth delay product of paths in order to avoid blocking.

59.2% Performance Improvement

Throughput (Mbps) vs Number of streams								
# Streams	**1**		**2**		**4**		**6**	
Queue Size	**1**	**1000**	**1**	**1000**	**1**	**1000**	**1**	**1000**
25-25 (ms)	291	291	580	580	580	584	580	585
50-25 (ms)	198	199	432	486	567	562	569	572
100-25 (ms)	97	96	304	380	452	478	524	541
200-25 (ms)	44	44	157	250	309	336	371	390
400-25 (ms)	19	19	60	122	139	209	183	278

*Note: when 1 stream is used it takes path of higher latency

Average Queue Length (Bytes) Comparing Paths of Varied Latency (2 Streams)									
25-25(ms)		50-25(ms)		100-25(ms)		200-25(ms)		400-25(ms)	
.00176	.00142	.00101	.04039	.00115	.32321	.01184	.623	.1411	.4805

Queue Yields Performance Increase When Path Latency Varies

- Receiver queue helps increase performance when multiple paths of varied latency are used.
- Need to have enough queue space to buffer the difference in bandwidth delay product of paths in order to avoid blocking.

103.3% Performance Improvement

Throughput (Mbps) vs Number of streams								
# Streams	**1**		**2**		**4**		**6**	
Queue Size	**1**	**1000**	**1**	**1000**	**1**	**1000**	**1**	**1000**
25-25 (ms)	291	291	580	580	580	584	580	585
50-25 (ms)	198	199	432	486	567	562	569	572
100-25 (ms)	97	96	304	380	452	478	524	541
200-25 (ms)	44	44	157	250	309	336	371	390
400-25 (ms)	19	19	60	122	139	209	183	278

*Note: when 1 stream is used it takes path of higher latency

Average Queue Length (Bytes) Comparing Paths of Varied Latency (2 Streams)									
25-25(ms)		50-25(ms)		100-25(ms)		200-25(ms)		400-25(ms)	
.00176	.00142	.00101	.04039	.00115	.32321	.01184	.623	.1411	.4805

Low Queue Utilization with Similar Path Latency

- Receiver queue helps increase performance when multiple paths of varied latency are used.
- Need to have enough queue space to buffer the difference in bandwidth delay product of paths in order to avoid blocking.

Throughput (Mbps) vs Number of streams								
# Streams	1		2		4		6	
Queue Size	1	1000	1	1000	1	1000	1	1000
25-25 (ms)	291	291	580	580	580	584	580	585
50-25 (ms)	198	199	432	486	567	562	569	572
100-25 (ms)	97	96	304	380	452	478	524	541
200-25 (ms)	44	44	157	250	309	336	371	390
400-25 (ms)	19	19	60	122	139	209	183	278

*Note: when 1 stream is used it takes path of higher latency

Not very utilized with similar latency

Average Queue Utilization(Bytes) Comparing Paths of Varied Latency (2 Streams)									
25-25(ms)		50-25(ms)		100-25(ms)		200-25(ms)		400-25(ms)	
.00176	.00142	.00101	.04039	.00115	.32321	.01184	.623	.1411	.4805

High Queue Utilization with Varied Path Latency

- Receiver queue helps increase performance when multiple paths of varied latency are used.
- Need to have enough queue space to buffer the difference in bandwidth delay product of paths in order to avoid blocking.

More utilized when varied path latency

Throughput (Mbps) vs Number of streams								
# Streams	**1**		**2**		**4**		**6**	
Queue Size	**1**	**1000**	**1**	**1000**	**1**	**1000**	**1**	**1000**
25-25 (ms)	291	291	580	580	580	584	580	585
50-25 (ms)	198	199	432	486	567	562	569	572
100-25 (ms)	97	96	304	380	452	478	524	541
200-25 (ms)	44	44	157	250	309	336	371	390
400-25 (ms)	19	19	60	122	139	209	183	278

*Note: when 1 stream is used it takes path of higher latency

Average Queue Utilization (Bytes) Comparing Paths of Varied Latency (2 Streams)									
25-25(ms)		50-25(ms)		100-25(ms)		200-25(ms)		400-25(ms)	
.00176	.00142	.00101	.04039	.00115	.32321	.01184	.623	.1411	.4805

Queue Size Effect with Multiple Paths of Varied Bandwidth

Throughput (Mbps) vs Number of Sockets Queue length 1					
	Path Bandwidth (Mbit)				
# Streams	100-100	200-100	300-100	400-100	500-100
1	95	190	199	198	199
2	190	240	240	238	238
4	192	267	268	272	267
6	193	261	230	198	279
8	171	221	230	260	239
10	195	200	194	236	204

Throughput (Mbps) vs Number of Sockets Queue length 10,000					
	Path Bandwidth (Mbit)				
# Streams	100-100	200-100	300-100	400-100	500-100
1	96	190	198	199	199
2	191	286	293	293	293
4	193	288	376	413	425
6	194	289	382	460	499
8	195	290	382	472	547
10	196	294	385	469	521

* When 1 stream is used, it runs over the path of higher available bandwidth.

Receiver Queue Provides Little Benefit using Paths of Same Bandwidth

.51% Performance Improvement

Throughput (Mbps) vs Number of Sockets Queue length 1					
	Path Bandwidth (Mbit)				
# Streams	100-100	200-100	300-100	400-100	500-100
1	95	190	199	198	199
2	190	240	240	238	238
4	192	267	268	272	267
6	193	261	230	198	279
8	171	221	230	260	239
10	195	200	194	236	204

Throughput (Mbps) vs Number of Sockets Queue length 10,000					
	Path Bandwidth (Mbit)				
# Streams	100-100	200-100	300-100	400-100	500-100
1	96	190	198	199	199
2	191	286	293	293	293
4	193	288	376	413	425
6	194	289	382	460	499
8	195	290	382	472	547
10	196	294	385	469	521

* When 1 stream is used, it runs over the path of higher available bandwidth.

Receiver Queue Provides Large Benefit using Paths of Same Bandwidth

47% Performance Improvement

Throughputs (Mbps) vs Number of Sockets Queue length 1					
	Path Bandwidth (Mbit)				
# Streams	100-100	200-100	300-100	400-100	500-100
1	95	190	199	198	199
2	190	240	240	238	238
4	192	267	268	272	267
6	193	261	230	198	279
8	171	221	230	260	239
10	195	200	194	236	204

Throughputs (Mbps) vs Number of Sockets Queue length 10,000					
	Path Bandwidth (Mbit)				
# Streams	100-100	200-100	300-100	400-100	500-100
1	96	190	198	199	199
2	191	286	293	293	293
4	193	288	376	413	425
6	194	289	382	460	499
8	195	290	382	472	547
10	196	294	385	469	521

* When 1 stream is used, it runs over the path of higher available bandwidth.

Receiver Queue Provides Large Benefit using Paths of Same Bandwidth

98.45% Performance Improvement

Throughput (Mbps) vs Number of Sockets Queue length 1					
	Path Bandwidth (Mbit)				
# Streams	100-100	200-100	300-100	400-100	500-100
1	95	190	199	198	199
2	190	240	240	238	238
4	192	267	268	272	267
6	193	261	230	198	279
8	171	221	230	260	239
10	195	200	194	236	204

Throughput (Mbps) vs Number of Sockets Queue length 10,000					
	Path Bandwidth (Mbit)				
# Streams	100-100	200-100	300-100	400-100	500-100
1	96	190	198	199	199
2	191	286	293	293	293
4	193	288	376	413	425
6	194	289	382	460	499
8	195	290	382	472	547
10	196	294	385	469	521

* When 1 stream is used, it runs over the path of higher available bandwidth.

Receiver Queue Provides Large Benefit using Paths of Same Bandwidth

98.73% Performance Improvement

Throughput (Mbps) vs Number of Sockets Queue length 1					
	Path Bandwidth (Mbit)				
# Streams	100-100	200-100	300-100	400-100	500-100
1	95	190	199	198	199
2	190	240	240	238	238
4	192	267	268	272	267
6	193	261	230	198	279
8	171	221	230	260	239
10	195	200	194	236	204

Throughput (Mbps) vs Number of Sockets Queue length 10,000					
	Path Bandwidth (Mbit)				
# Streams	100-100	200-100	300-100	400-100	500-100
1	96	190	198	199	199
2	191	286	293	293	293
4	193	288	376	413	425
6	194	289	382	460	499
8	195	290	382	472	547
10	196	294	385	469	521

* When 1 stream is used, it runs over the path of higher available bandwidth.

Receiver Queue Provides Large Benefit using Paths of Same Bandwidth

155.4% Performance Improvement

Throughput (Mbps) vs Number of Sockets Queue length 1					
	Path Bandwidth (Mbit)				
# Streams	**100-100**	**200-100**	**300-100**	**400-100**	**500-100**
1	95	190	199	198	199
2	190	240	240	238	238
4	192	267	268	272	267
6	193	261	230	198	279
8	171	221	230	260	239
10	195	200	194	236	204

Throughput (Mbps) vs Number of Sockets Queue length 10,000					
	Path Bandwidth (Mbit)				
# Streams	**100-100**	**200-100**	**300-100**	**400-100**	**500-100**
1	96	190	198	199	199
2	191	286	293	293	293
4	193	288	376	413	425
6	194	289	382	460	499
8	195	290	382	472	547
10	196	294	385	469	521

* When 1 stream is used, it runs over the path of higher available bandwidth.

Lossy Link

- SOS is able to overcome a lossy link using multiple streams coupled with user space Queuing.

1% Lossy Path vs Nonlossy Path Throughput (Mbps)				
# Sockets	Lossy Path Throughput (Queue Size 1)	Lossy Path Throughput (Queue Size 1,000)	Nonlossy Path Throughput Mbps (Queue Size 1)	Nonlossy Path Throughput Mbps (Queue Size 1,000)
1	16	29	805	897
2	30	32	902	877
4	51	90	900	904
6	93	110	884	899
8	93	124	902	902
10	121	136	883	888
20	135	149	902	904
30	213	316	905	906
40	251	484	902	904
50	334	632	903	902
60	348	740	905	901
70	392	796	906	897
80	415	865	905	897
90	404	867	902	894
100	448	874	902	892

Lossy Link Obtains High Throughput using Many Streams

- SOS is able to overcome a lossy link using multiple streams coupled with user space Queuing.

2,913.8% Performance Improvement

1% Lossy Path vs Nonlossy Path Throughput (Mbps)				
# Sockets	Lossy Path Throughput (Queue Size 1)	Lossy Path Throughput (Queue Size 1,000)	Nonlossy Path Throughput Mbps (Queue Size 1)	Nonlossy Path Throughput Mbps (Queue Size 1,000)
1	16	29	805	897
2	30	32	902	877
4	51	90	900	904
6	93	110	884	899
8	93	124	902	902
10	121	136	883	888
20	135	149	902	904
30	213	316	905	906
40	251	484	902	904
50	334	632	903	902
60	348	740	905	901
70	392	796	906	897
80	415	865	905	897
90	404	867	902	894
100	448	874	902	892

Lossy Link Achieves Similar throughput as Nonlossy Link

- SOS is able to overcome a lossy link using multiple streams coupled with user space Queuing.

2.6% Performance Difference

1% Lossy Path vs Nonlossy Path Throughput (Mbps)				
# Sockets	Lossy Path Throughput (Queue Size 1)	Lossy Path Throughput (Queue Size 1,000)	Nonlossy Path Throughput Mbps (Queue Size 1)	Nonlossy Path Throughput Mbps (Queue Size 1,000)
1	16	29	805	897
2	30	32	902	877
4	51	90	900	904
6	93	110	884	899
8	93	124	902	902
10	121	136	883	888
20	135	149	902	904
30	213	316	905	906
40	251	484	902	904
50	334	632	903	902
60	348	740	905	901
70	392	796	906	897
80	415	865	905	897
90	404	867	902	894
100	448	874	902	892

Sending/Receiving Buffer Size Effect

- The number of bytes agent should try to send/receive each time was found to greatly affect performance.
  - Send(fd, buf, sizeof(buf), 0);

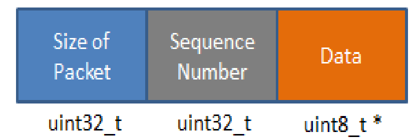
- Initial thought was due to overhead added by additional headers.

27.7% Performance Improvement

Varying Buffer Size Results (100ms RTT)					
Buffer Size (Bytes)	**512**	**1024**	**2048**	**4096**	**8192**
Throughput (Mbps)	155	156	169	188	198
# Application Level Packets	2311930	1152818	622046	345238	182105
% Overhead	1.56	0.78	0.39	0.195	0.097
Average Packet Size Send/Received (Bytes)	511.99	1023.99	2047.99	4095.97	8195.92

SOS Overhead

- Addition 8 bytes was added in order to breakup and reassemble data
- Size of packet and sequence number
- 8/512 = 1.56%
- 8/8196 = .0976%
- Not always true do to amount available to read



Accounting for overhead:

Sends Block size – Header packets
(2311930*(511.99 -8)*8)/(60*10^6)
: **155.35Mbps**

Assuming no overhead
(2311930*(511.99)*8)/(60*10^6)
: 157.82Mbps

: 2.46Mbps Lost due to overhead

Varying Block Size Results (100ms RTT)					
Block Size (Bytes)	**512**	**1024**	**2048**	**4096**	**8192**
Throughput (Mbps)	155	156	169	188	198
# Application Level Packets	2311930	1152818	622046	345238	182105
% Overhead	1.56	0.78	0.39	0.195	0.098
Average Packet Size Send/Received (Bytes)	511.99	1023.99	2047.99	4095.97	8195.92

Missing Throughput

- **2.46Mbps Lost due to overhead**

- **Though, 198-157.82 = 40.18Mbps Where did this go?**

Varying Block Size Results (100ms RTT)					
Buffer Size (Bytes)	**512**	**1024**	**2048**	**4096**	**8192**
Throughput (Mbps)	155	156	169	188	198
# Application Level Packets	2311930	1152818	622046	345238	182105
% Overhead	1.56	0.78	0.39	0.195	0.098
Average Packet Size Send/Received (Bytes)	511.99	1023.99	2047.99	4095.97	8195.92

Maybe due to TCP Overhead?

- Lower Average Segment Size when using smaller buffer
- Requires more packets (+54 Byte header per packet)
- Values obtained via tcptrace which analysis's the pcap trace

Average Segment Size vs Block Size (100ms RTT)					
Buffer Size (bytes)	**512**	**1024**	**2048**	**4096**	**8192**
Average Segment Size (bytes)	1493	1494	2107	4193	8354
# Packets Sent	792,416	789,710	604,612	337,199	178,640

343.58% More packets sent, though 27.7% slower!

Maybe due to TCP Overhead?

- Lower Average Segment Size when using smaller buffer
- Requires more packets (+54 Byte header per packet)
- Values obtained via tcptrace which analysis's the pcap trace

Average Segment Size vs Block Size (100ms RTT)					
Buffer Size (bytes)	**512**	**1024**	**2048**	**4096**	**8192**
Average Segment Size (bytes)	1493	1494	2107	4193	8354
# Packets Sent	792,416	789,710	604,612	337,199	178,640

Why is this bigger than my 1500MTU?

TCP Segmentation Offload (TSO)

- Offload Segmentation to NIC
  - Can pass down 64KB of data to NIC which breaks into MTU chunks
  - Drastically reduces CPU time needed to transmit data
  - Segment Size does come into play with high throughput

Average Segment Size vs Block Size (100ms RTT)					
Block Size (bytes)	**512**	**1024**	**2048**	**4096**	**8192**
Thoughput (Mbps)	155	157	157	158	159
Average Segment Size (bytes)	1447	1447	1447	1447	1447

Average Segment Size vs Block Size (0ms RTT)					
Block Size (bytes)	**512**	**1024**	**2048**	**4096**	**8192**
Throughput (Mbps)	464 (816)	587 (824)	826 (852)	863 (893)	870 (909)
Average Segment Size (bytes)	1335	1392	1447	1447	1447

Improving Performance

- No one magic number for amount for number of sockets to use.
- What information can SOS use to improve performance?
- Queue Utilization, Average sent/received block size
- Rexmt is the number of retransmitted data

# Streams	Throughput (Mbps)	Average Received Block Size (Bytes)	Average Queue Utilization	Rexmt (MB)	Sent Data in 5 minutes (GB)
5	597	8180.2	.07026	0	20.78
100	512	7217.98	.62778	3.4	17.94

Improving Performance

- Using the average receive block size and average queue utilization agent could detect over utilization and disable sockets to improve performance

# Streams	Throughput (Mbps)	Average Received Block Size (Bytes)	Average Queue Utilization	Rexmt (MB)	Sent Data in 5 minutes (GB)
5	597	8180.2	.07026	0	20.78
100	512	7217.98	.62778	3.4	17.94

Signs of over utilization

Improving Performance

- Multipath example with varied available bandwidth.
- 20 streams used which achieved throughput 111Mbps
- Agent spent most of time blocking on streams that traversed 300Mbit path.

Path	Average Block Size(Bytes)	Average Queue Utilization
20Mbit	8181.1	.13529
300Mbit	8168.3	.92177

- Agent could detect this and dynamically alter the number of steams that traversed each path to improve performance.
- Not Implemented nor evaluated.

Conclusion

- SOS shows how network providers can decouple users choice of protocols to network providers choice of protocols.
- Alleviates complexes for end user since no modifications are required.
- Throughput improvements are shown over high Delay Bandwidth Product Networks and Lossy links
- Receiver Queue significantly helps to improve throughput results in multipath scenarios.

Future Work

- Improve SOS to dynamically alter number of sockets used in order to optimize throughput
- Develop other services such as client mobility, delay tolerant network services, and increased security could be provided via similar methods