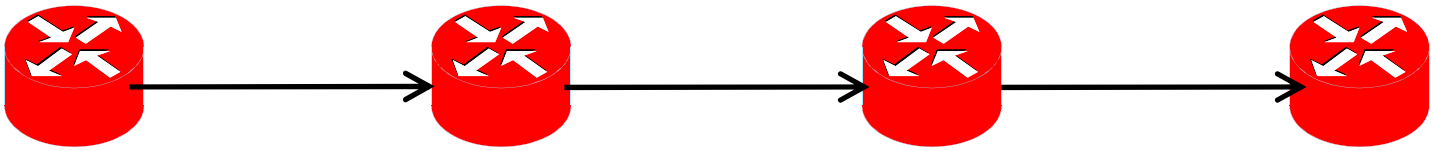
Module 4:
Data Plane

- Software Based Data Plane : Click
- Hardware Based Data Plane : NetFPGA
- Network Programming Language
- Composition of SDN

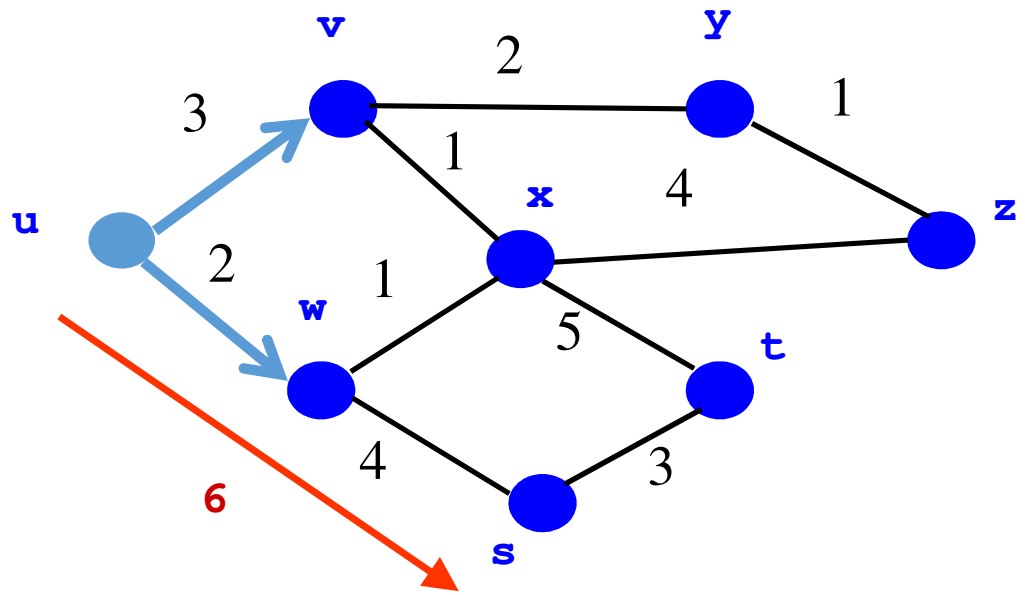
Forwarding Vs. Routing

- Forwarding: data plane
  - Directing a data packet to an outgoing link
  - Individual router *using* a forwarding table
- Routing: control plane
  - Computing paths the packets will follow
  - Routers talking amongst themselves
  - Individual router ***creating*** a forwarding table



Example: Shortest-Path Routing

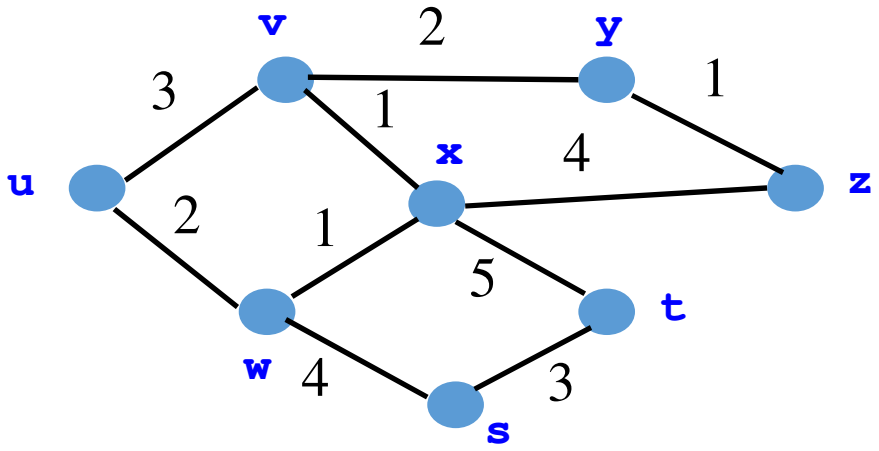
- Compute: *path costs* to all nodes (single control plane)
  - From a source u to all other nodes
  - Cost of the path through each link
  - Next hop along least-cost path to s



	link
v	(u,v)
w	(u,w)
x	(u,w)
y	(u,v)
z	(u,v)
s	(u,w)
t	(u,w)

Distributed Control Plane

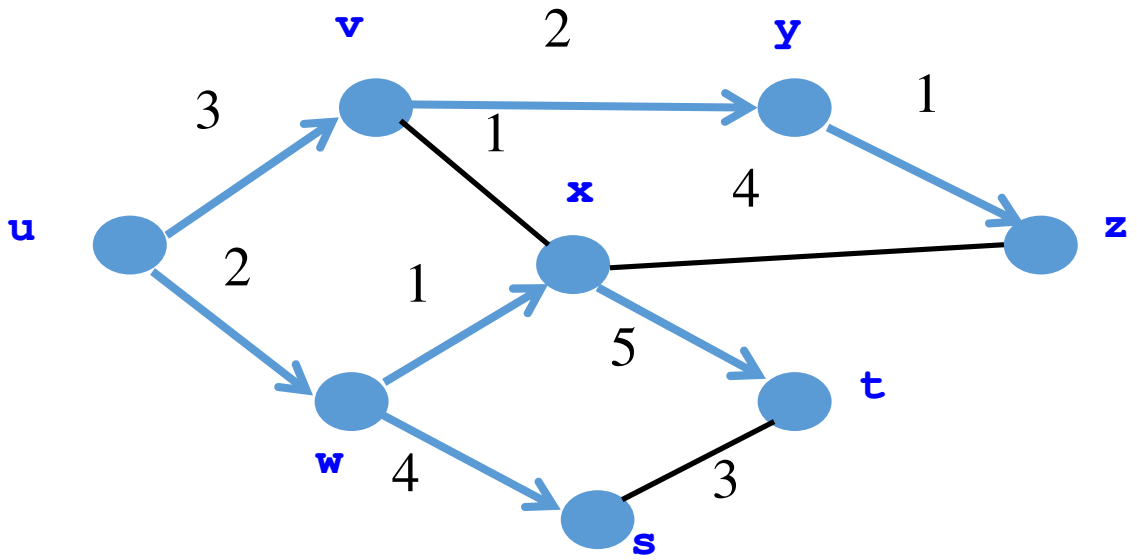
- Distance-vector routing: RIP, EIGRP
  - Each node computes path cost
  - … based on each neighbors' path cost
  - Bellman-Ford algorithm



$$d_u(z) = \min\{c(u,v) + d_v(z), \; c(u,w) + d_w(z)\}$$

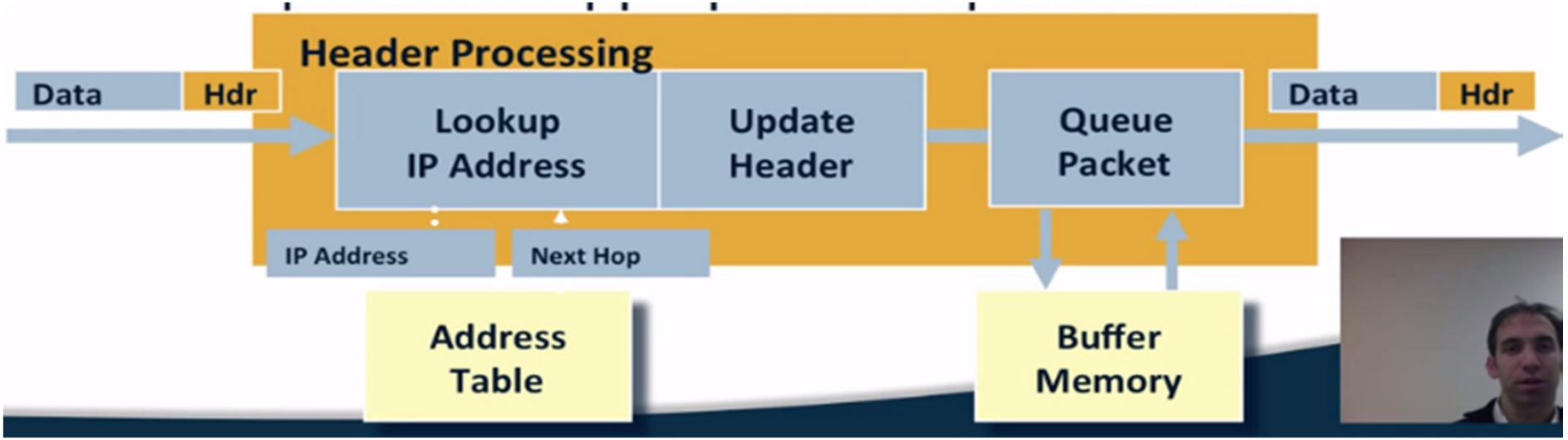
Distributed Control Plane

- Link-state routing: OSPF, IS-IS
  - Flood the entire topology to all nodes
  - Each node computes shortest paths



Review of Data Plane:

- Router gets packet
- Looks at packet header for destination
- Looks up forwarding (flow) table for output interface
- Modifies header (TTL, IP header checksum)
- Passes packet to appropriate output interface



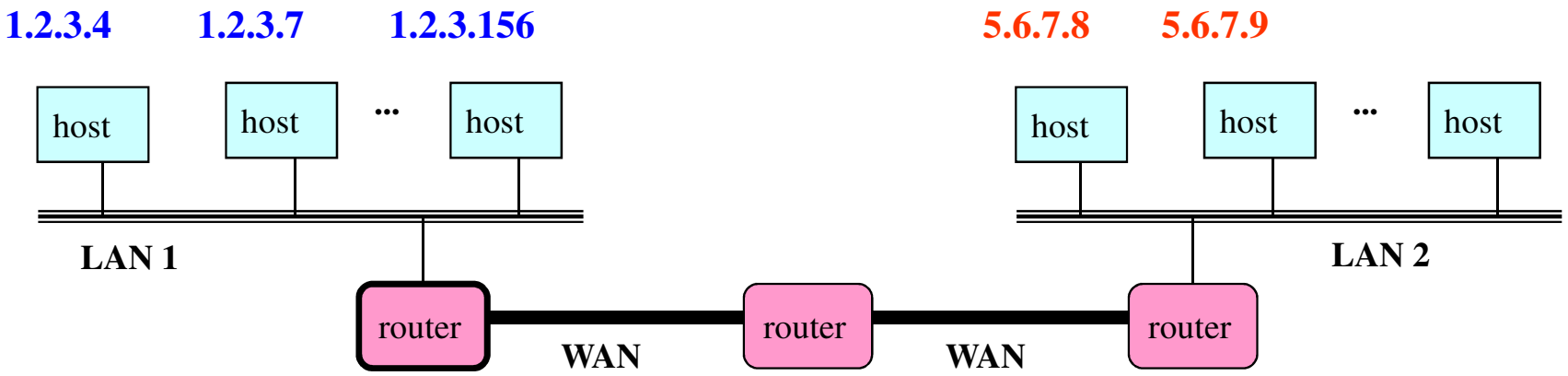
- Data plane operations typically includes:
  - Streaming algorithms that act on packets
  - Matching some bits and taking a simple action
- Primitive is *<Match, Action>*
- ***Match*** arbitrary bits in headers:

  - Match on any header, or new header
  - Allows any flow granularity
- ***Action***
  - Forward to port(s), drop, send to controller
  - Overwrite header with mask, push or pop
  - Forward at specific bit-rate

- Wide range of data plane's functions:
  1. Forwarding
  2. Access Control
  3. Mapping Header Fields
  4. Traffic Engineering
  5. Buffering & Marking
  6. Shaping & Scheduling
  7. Deep Packet Inspection

1. Forwarding:

- IP Forwarding

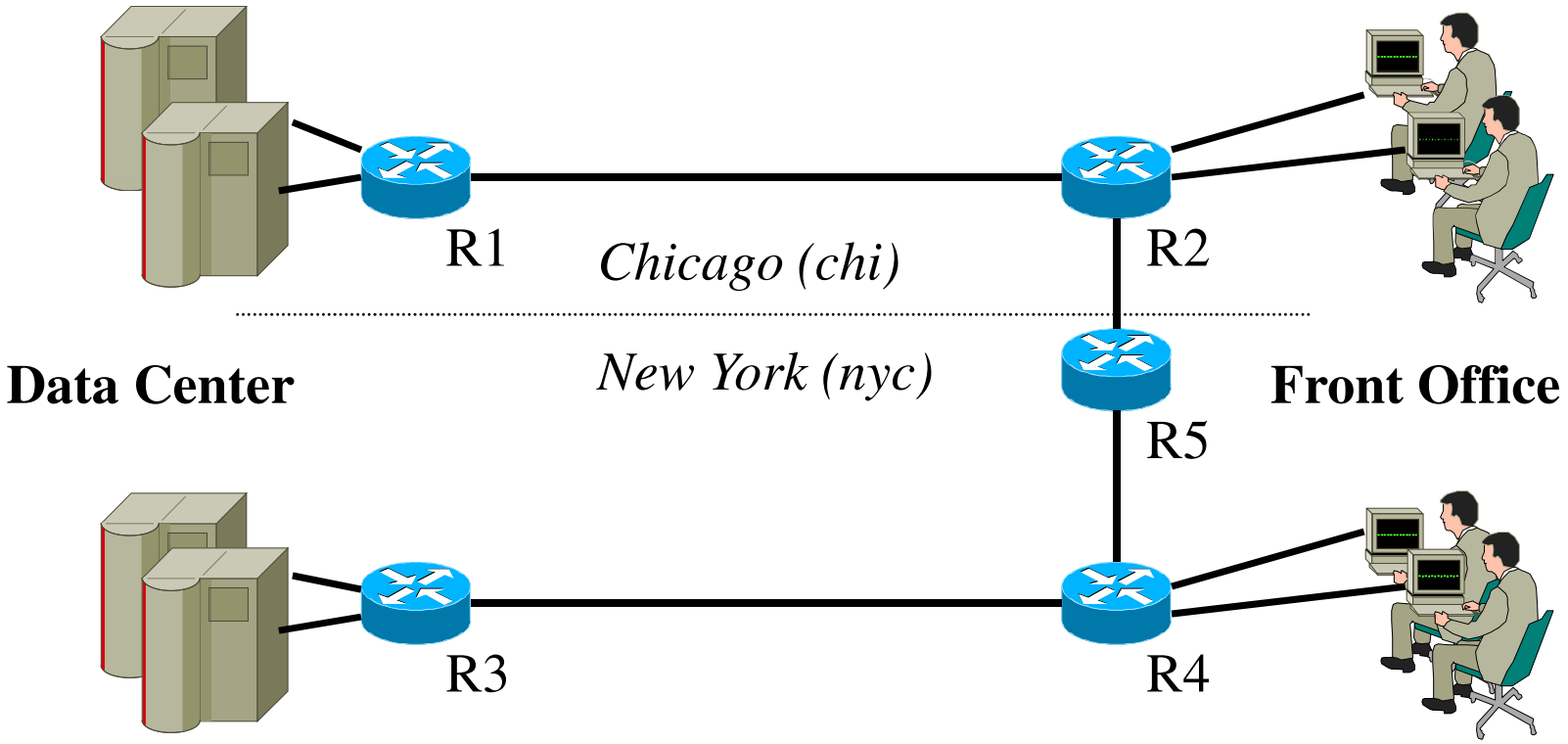


1.2.3.0/24	←
5.6.7.0/24	→

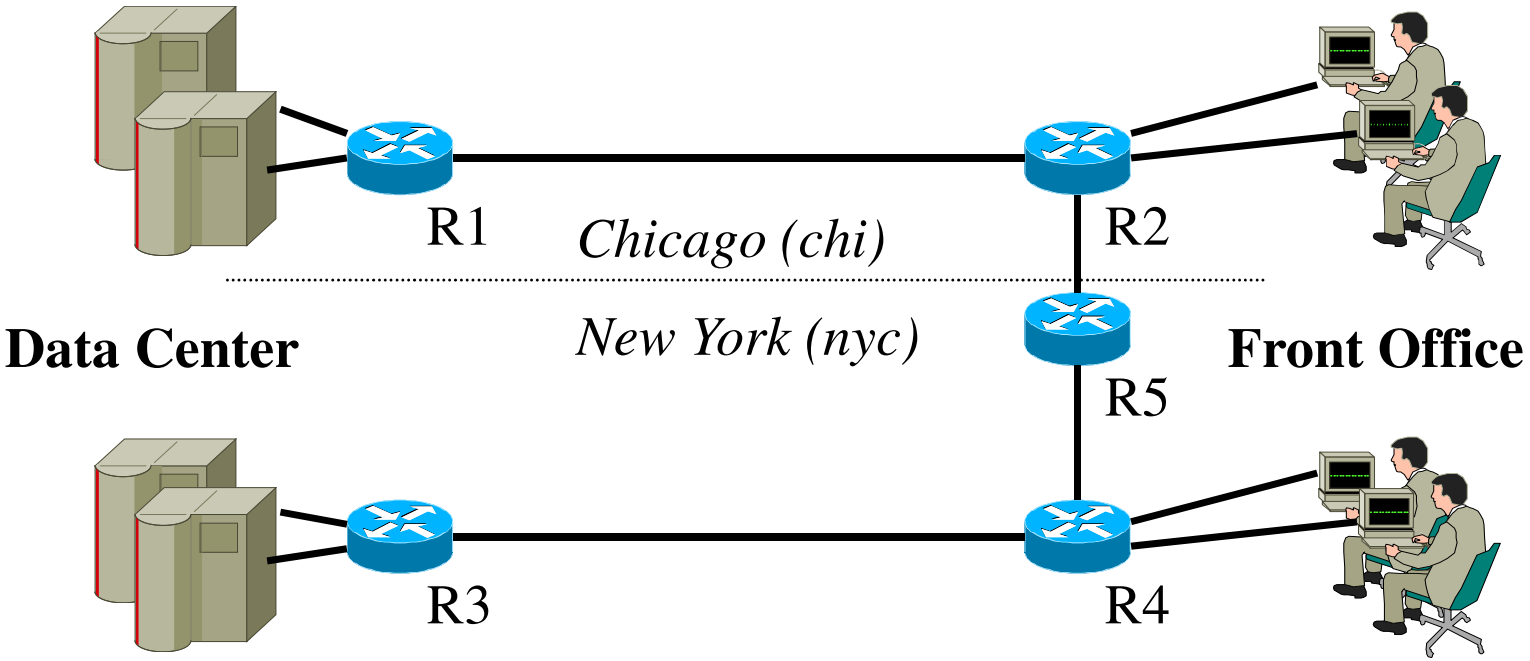
forwarding table

2. Access Control

- Two locations, each with data center & front office
- All routers exchange routes over all links



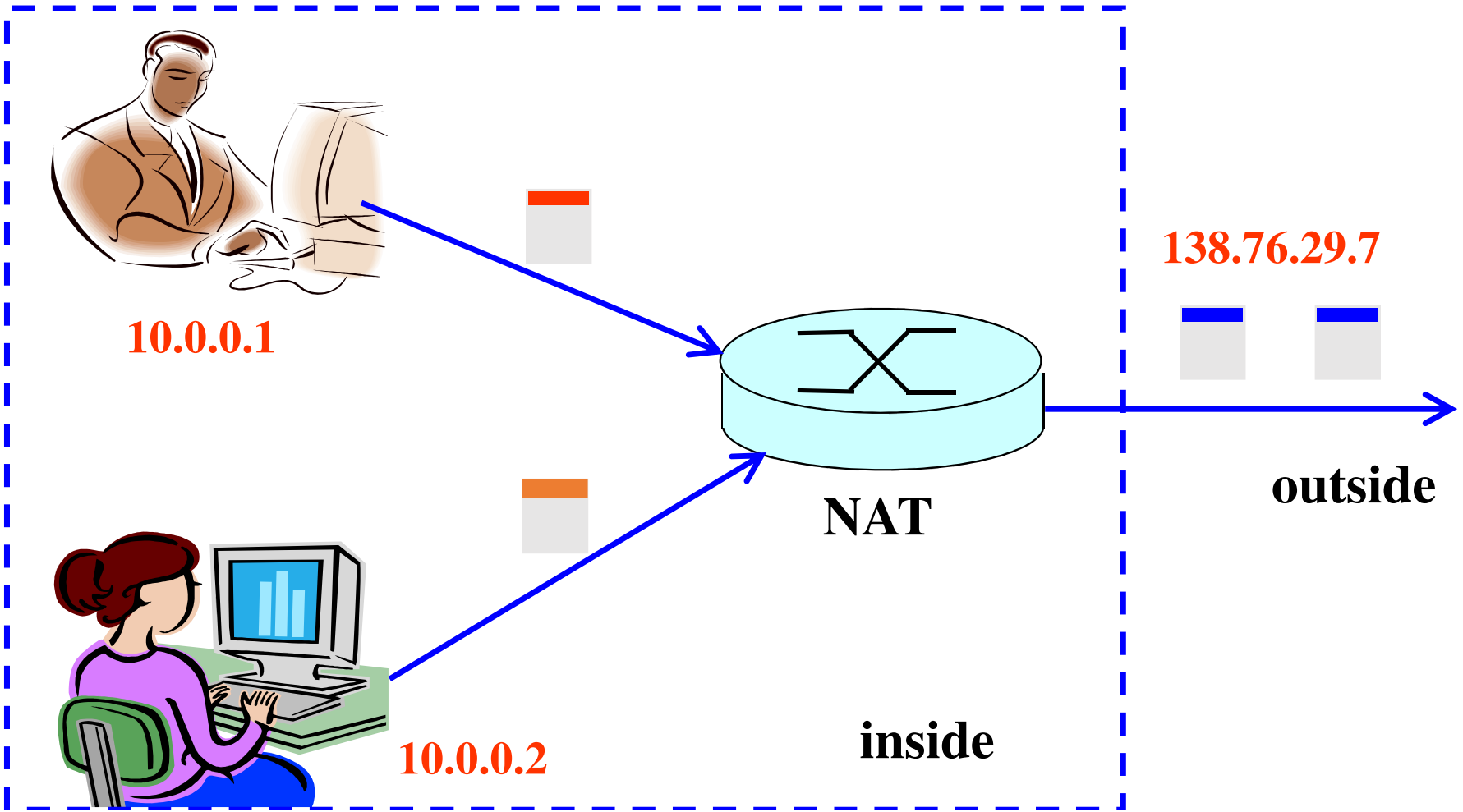
2. Access Control:



	chi-DC	chi-FO	nyc-DC	nyc-FO
chi-DC		●	●	⊘
chi-FO	●		⊘	●
nyc-DC	●	⊘		●
nyc-FO	⊘	●	●	

3. Mapping Header Fields

3.1 Network Address Translation (NAT)

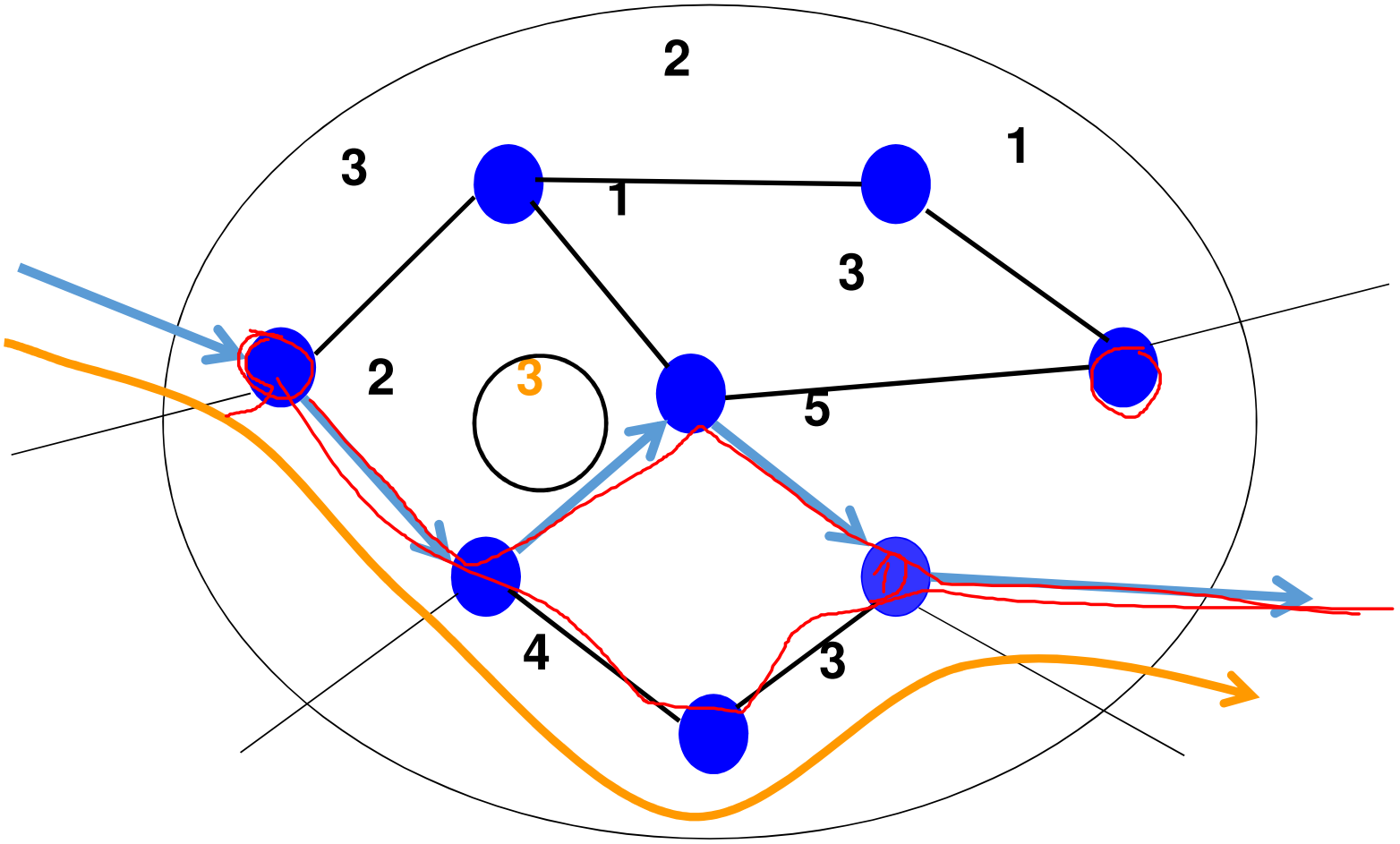


3.2 Mapping Addresses and Ports

- Remap IP addresses and TCP/UDP port numbers
  - Addresses: between end-host and NAT addresses
  - Port numbers: to ensure each connection is unique
- Create table entries as packets arrive
  - Src 10.0.0.1, Sport 1024, Dest 1.2.3.4, Dport 80
    - Map to Src 138.76.29.7, Sport 1024, Dest 1.2.3.4, Dport 80
  - Src 10.0.0.2, Sport 1024, Dest 1.2.3.4, Dport 80
    - Map to Src 138.76.29.7, Sport 1025, Dest 1.2.3.4, Dport 80
- Challenges
  - When to remove the entries
  - Running services behind a NAT
  - What if both ends of a connection are behind NATs

4. Traffic Engineering:

- Traffic engineering is a method of optimizing the performance of a network by dynamically analyzing, predicting and regulating the behavior of data transmitted over a network.
- Which paths to use to deliver traffic?
- How to control paths?
  - Set link weights used by routing protocol


2
3
1
1
3
2
3
5
4
3

- In a congested network, one of three things can happen when a subscriber attempts to send a message or place a call:
  - The user receives a busy signal or other indication that the network cannot carry out a call at that time.
  - A message is placed in a queue and is eventually delivered according to specified parameters.
  - A message is rejected, returned or lost.
- When message queues become unacceptably long or the frequency of busy signals becomes unacceptably high, the network is said to be in a **high-loss condition.**

- Objective of traffic engineering:
  1. To minimize or eliminate high-loss situations. In particular, the number of rejected messages or failed call attempts should be as close to zero as possible.
  2. To balance the QoS against the cost of operating and maintaining the network.
- In SDN it is difficult to handle traffic engineering because links are not static, they are dynamic as they are configured with the help of software.

Technique	Description	Routing	Comments
B4 (by Google)	• It uses a centralized TE, layered on top of the routing protocols • To achieve fairness it allocates resources using Min-Max fairness technique.	• It uses hashed-based ECMP to balance the load among multiple links.	•If TE service can be stopped so that the packets are forwarded using short path forwarding mechanism.
Hedera	• detects the elephant-flows at the edge switches, • if threshold is met, i.e. 10% of NIC bandwidth, the flow is marked as elephant flow,	• uses the global view of network and calculate the better paths, which are non-conflicting, for the elephant flows.	• achieves better optimal bisection of bandwidth of network
MicroTE	• detect the elephant flows at end host • Mainly designed for data centers	• uses short term predictability to route the traffic on multiple paths	• if traffic is predictable it perform close to optimal performance

5. Buffering & Marking:

- Buffer is a chunk of memory that stores packets temporarily when there is too much data sent to a network interface.
- Packet marking needed for router to **distinguish** between different classes;
- and also new router policy to treat packets accordingly.

6. Shaping & Scheduling:
7. Deep Packet Inspection :

Deep packet inspection (DPI) is an advanced method of examining and managing network traffic.

It is a form of packet filtering that locates, identifies, classifies, reroutes or blocks packets with specific data or code payloads that conventional packet filtering, which examines only packet headers, cannot detect.

Programmable/ Software Data Plane

- **CLICK : MODULAR ROUTER**
  - It is a new software architecture for building flexible and configurable routers.
  - It is assembled from packet processing modules called elements.
  - Individual elements implement simple router functions like packet classification, queuing, scheduling, and interfacing with network devices.
  - A router configuration is a directed graph with elements at the vertices; packets flow along the edges of the graph.

Click Motivation

- Flexibility
  - Add new features and enable experimentation
- Openness
  - Allow users/researchers to build and extend
  - (In contrast to most commercial routers)
- Modularity
  - Simplify the composition of existing features
  - Simplify the addition of new features
- Speed/efficiency
  - Operation (optionally) in the operating system
  - Without user needing to grapple with OS internals

- Several features make individual elements more powerful and complex configurations easier to write, including pull connections, which model packet flow driven by transmitting hardware devices, and flow-based router context, which helps an element locate other interesting elements.

Why there is a need of Click ?

- Boundary routers must often do:
  1. Prioritize traffic
  2. Translate Network addresses
  3. Tunnel and filter packets
  4. Act as firewall
- Network Administrator (NA):
  - May be able to turn router functions on or off.
  - Can’t easily specify / identify interactions of different functions.
  - It is difficult for NA or third party s/w vendor to extend a router with new functions

CLICK

- It is a flexible, modular software architecture for creating routers.
- Built from fine-grained components.
- Components are packet processing modules called elements.
- To build a router user chooses collection of elements and connects them into a directed graph.
- To extend a configuration, user can write new elements or compose existing elements in new ways.(openness)

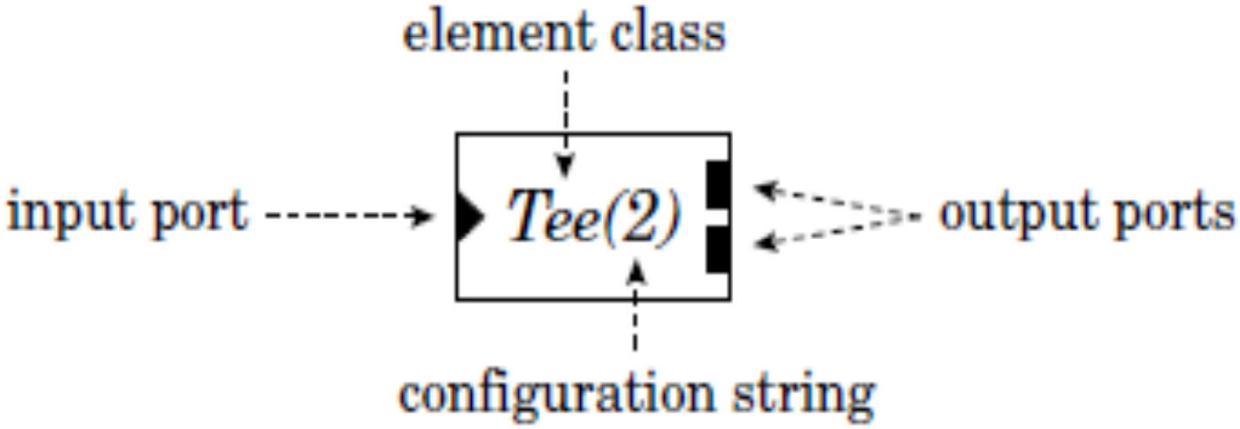
Aspects of CLICK Architecture

1. Packet handoff along a connection may be initiated by either source end (push) or destination end (pull)
2. A flow based router context mechanism lets an element automatically locate other elements on which it depends; it is based on the observation that relevant elements are often connected by the flow of packets.

- These features make individual elements more powerful and configurations easier to write.

A. CLICK ARCHITECTURE

- Click element represents a unit of router processing
- An element represents a simple computation, such as decrementing an IP packet's time to-live field, rather than a large, complex computation, such as IP routing.
- The user determines what a click router does by choosing the elements to be used and connections among them.
- Inside a running router:
  - Each element is a C++ object that may maintain a private state.
  - Connections are pointers to element objects.
  - Passing a packet along a connection is implemented as a single virtual function call.



A sample element. Triangular ports are inputs and rectangular ports are outputs.

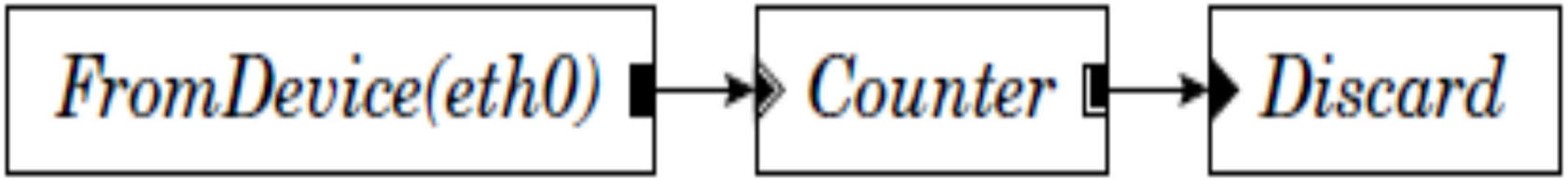


Fig. 2. A router configuration that throws away all packets.

A1. Properties of an element:

1. **Element Class:**
   - Each element belongs to 1 class.
   - It consists of code that should be executed when element processes a packet
   - Also elements initialization procedure and data layout.
2. **Ports:**
   - Element can have any number of input & output ports.
   - Every connection goes from an o/p port of one element to i/p port of another.
   - Different ports may have different functionalities.

3. Configuration String:

- It is optional
- Contains additional arguments that are passed to the element at the time of router initialization.

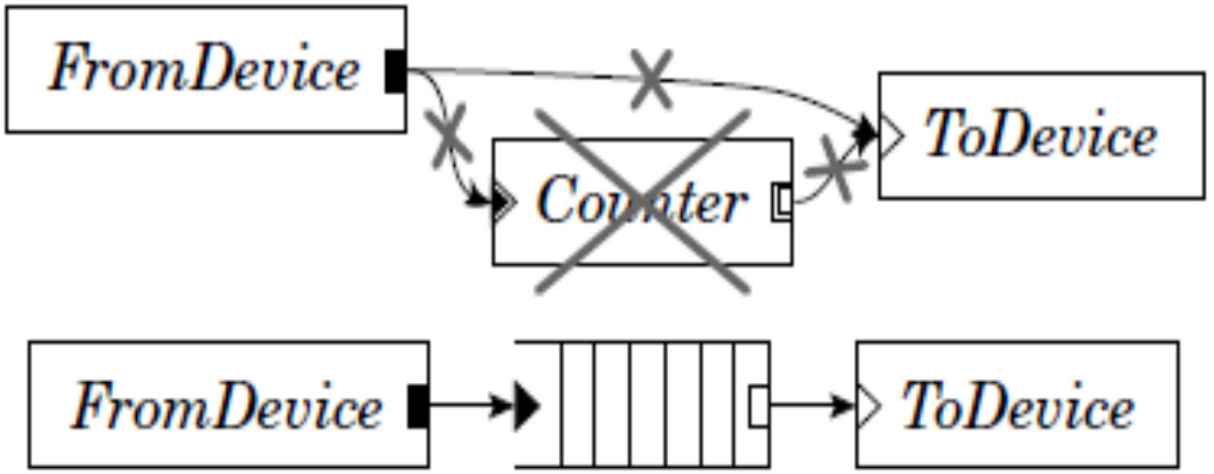
4. Method Interfaces:

- Each element support one or more method interfaces.
- Every element supports the simple packet transfer interface.
- Elements can create and export arbitrary additional interfaces.
- Elements communicate through these interfaces at run time which contain both data and methods.

A2. Push & Pull Connections:

- Click supports 2 kinds of connections, push & pull.
- On a push connection, packets start at the source element and are passed downstream to the destination element.
- On a pull connection, in contrast, the destination element initiates packet transfer: it asks the source element to return a packet, or a null pointer if no packet is available. (Clark – called upcalls).
- The processing type of a connection - whether it is push or pull - is determined by the ports at its endpoints.
- Each port in a running router is either push or pull;
- Connections between two push ports are push, and connections between two pull ports are pull.

- Connections between a push port and a pull port are illegal.
- Elements set their ports' types as the router is initialized.
- They may also create agnostic ports, which behave as push when connected to push ports and pull when connected to pull ports.
- When a router is initialized, the system propagates constraints until every agnostic port has been assigned to either push or pull.
- Diagram shown below is not properly configured.



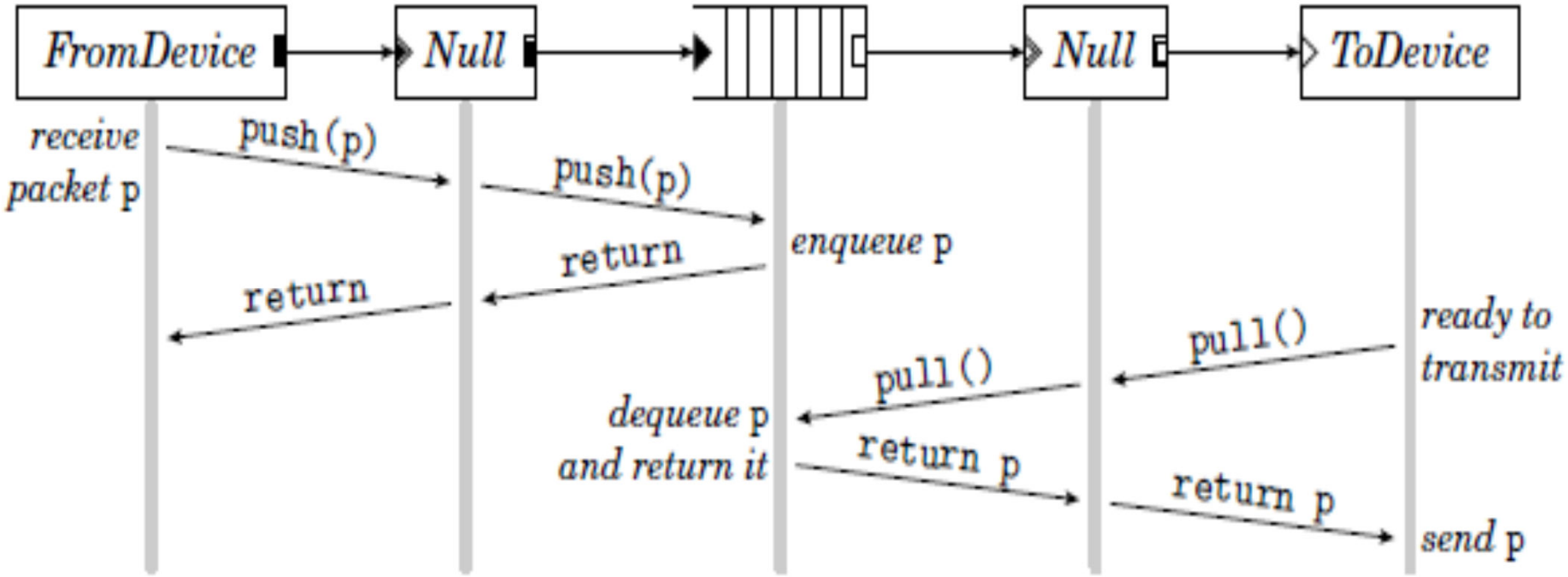


Fig. 3. Push and pull control flow. This diagram shows functions called as a packet moves through a simple router; time moves downwards. The central element is a *Queue*. During the push, control flow moves forward through the element graph starting at the receiving device; during the pull, control flow moves backward through the graph, starting at the transmitting device. The packet p always moves forward.

A3. Packet Storage

- Click elements don’t have implicit queues for storing packets.
- They are explicit objects, implemented a separate queue element.
- It also enables valuable configurations that are difficult to arrange, e.g. a single queue feeding multiple devices, or a queue feeding traffic shaper element.
- These explicit queues require both push and pull connections at input and output port respectively.
- Input port responds to pushed packets (push) by enqueueing them.
- Output port responds to pull requests by dequeueing packets and returning them.

A4. Language

- Click cogurations are written in a simple language with two important constructs:
  1. *Declarations:* create elements,
  2. *Connections:* say how they should be connected.

```
// Declare three elements ...
src :: FromDevice(eth0);
ctr :: Counter;
sink :: Discard;
// ...and connect them together
src -> ctr;
ctr -> sink;

// Alternate definition using syntactic sugar
FromDevice(eth0) -> Counter -> Discard;
```

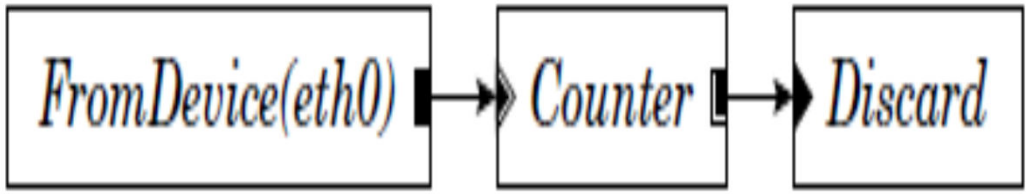
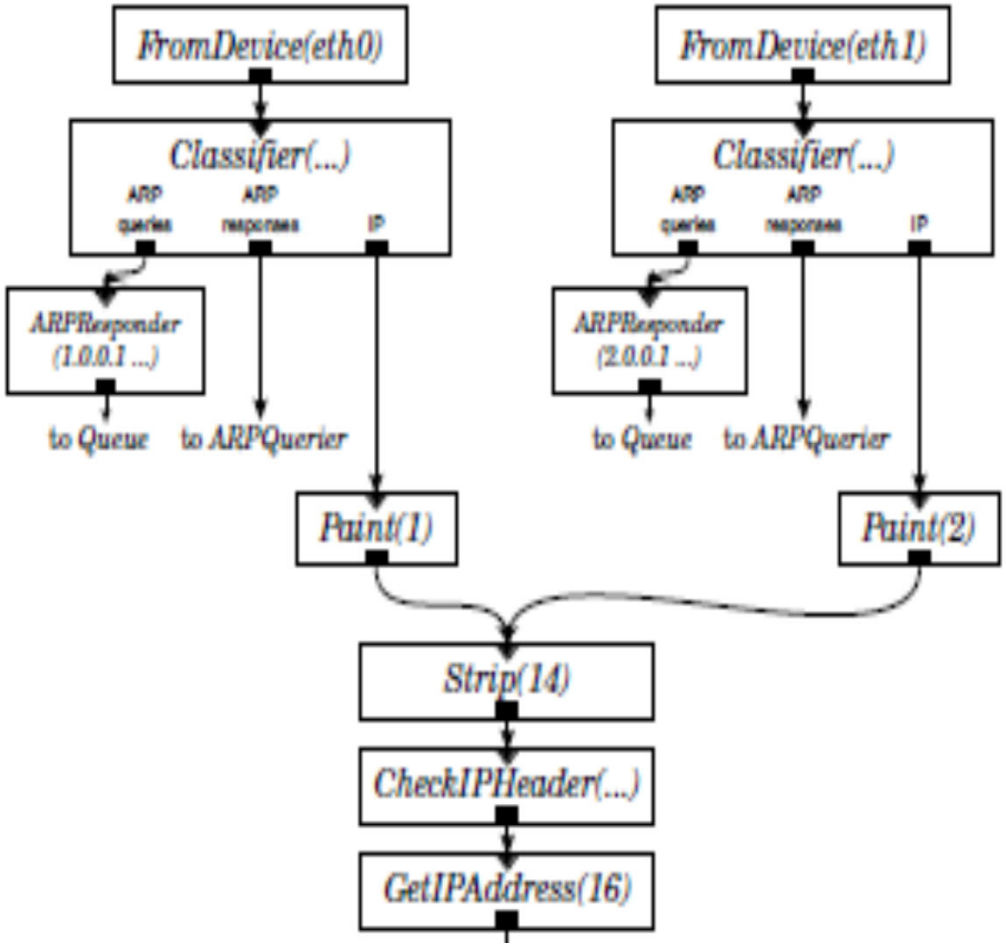


Fig. 2. A router configuration that throws away all packets.

- Language also contains compound elements, that lets users define their own element classes.
- A compound element is a router configuration fragment that behaves like an element class.
- A user could define a compound element consisting of a Queue followed by a Shaper, and call the resulting element class ShapedQueue;
- The rest of the configuration could then use ShapedQueue as if it were a primitive Click element class.
- Compound elements can have multiple ports with arbitrary push/pull characteristics.

A5. Element Design & Architectural Limitations

- many configurations only require per-flow CPU scheduling,
- Unlike flow-based systems, Click cannot schedule the CPU per individual flow.
- The Click language contains no notion of configuration variables.
- For example, it would be useful to refer to \the Ethernet address of the device handled by that FromDevice element" in a configuration string. Currently, the user must copy such an Ethernet address everywhere it is required, which leads to duplication of information.
- A configuration-variable extension would be simple to add.


FromDevice(eth0)
FromDevice(eth1)
Classifier(...)
ARP queries
ARP responses
IP
Classifier(...)
ARP queries
ARP responses
IP
ARPResponder (1.0.0.1 ...)
ARPResponder (2.0.0.1 ...)
to Queue
to ARPQuerier
to Queue
to ARPQuerier
Paint(1)
Paint(2)
Strip(14)
CheckIPHeader(...)
GetIPAddress(16)

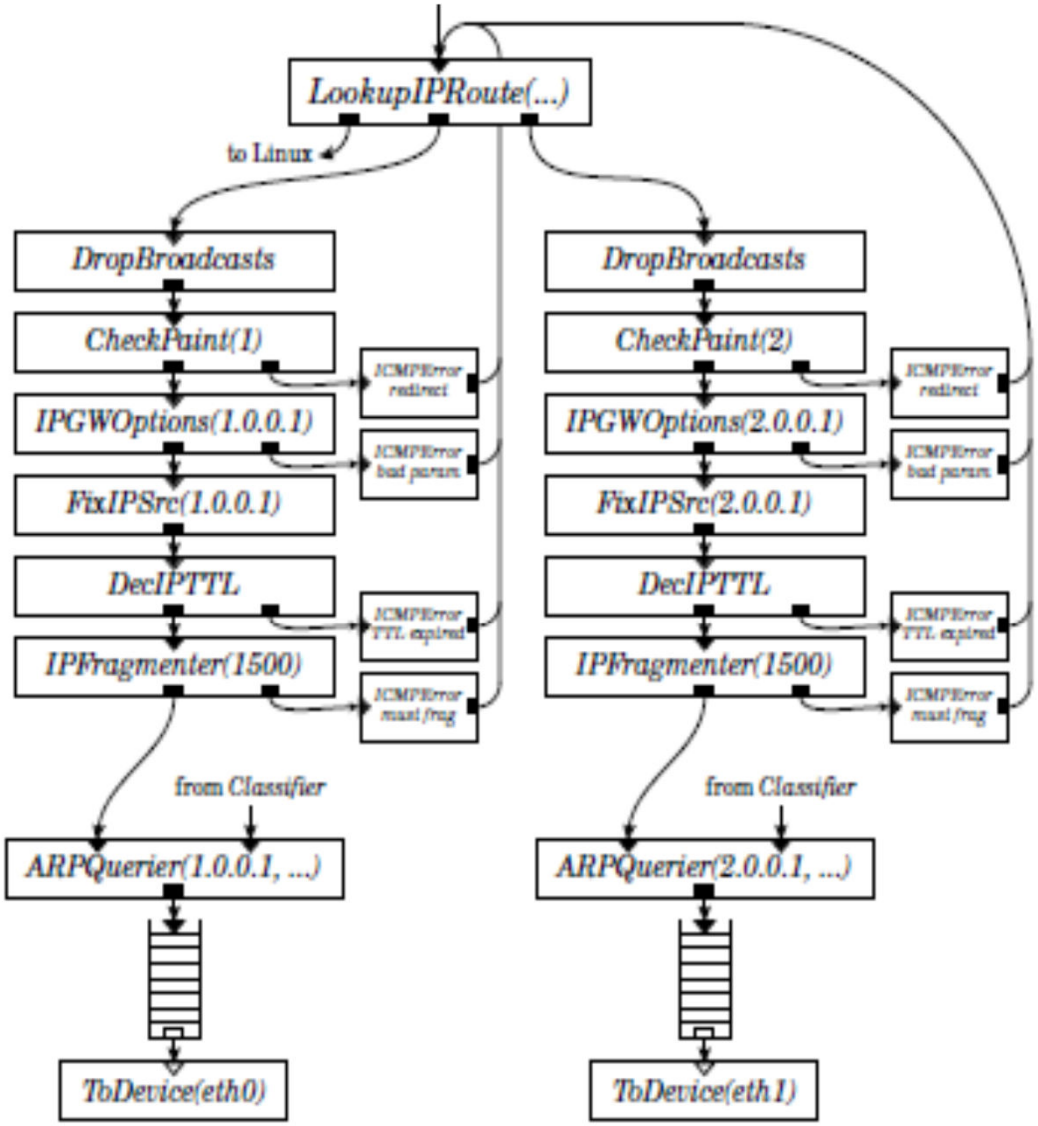


Fig. 8. An IP router configuration.