Algorithms for Software-Defined Distributed Systems

Stefan Schmid
TU Berlin & Telekom Innovation Labs (T-Labs)
Flexible Distributed Systems: Programmable...

SDN outsources and consolidates control over multiple devices to (logically) centralized software controller.
Flexible Distributed Systems: Programmable...

Benefit 1: Decoupling! Control plane can **evolve independently** of data plane: innovation at speed of software development.

(logically) centralized software controller

Benefit 2: Simpler network management through logically **centralized view**: network management is an inherently **non-local** task. Simplified **formal** verification.
SDN outsources and consolidates control over multiple devices to (logically) centralized software controller.

**Benefit 3:** Standard API OpenFlow is about **generalization**!
- Generalize **devices** (L2-L4: switches, routers, middleboxes)
- Generalize **routing and traffic engineering** (not only destination-based)
- Generalize **flow-installation**: coarse-grained rules and wildcards okay, proactive vs reactive installation
- Provide general and logical **network views** to the application / tenant
Flexible Distributed Systems: ... and Virtualized

- Virtualization allows to **abstract**:
  - Hardware: compute, memory, storage, network resources
  - Or even entire distributed systems (including OS)

- **Decouples** the application from the substrate

- Introduces **flexibilities** for resource allocation
  - Improved **resource sharing** (esp. in commercial clouds)
  - Seamless migration
Challenges

- Great..., but: SDN and virtualization are enablers, not solutions! What to do with them and how?

- Example: Virtualization for better resource sharing
  - Many flexibilities to embed virtual machines
  - But: often not enough to provide the expected performance!

Need to virtualize the entire system: otherwise risk of interference on other resources (network, CPU, memory, I/O): unpredictable performance.
For predictable performance: full virtualization!

App 1: Mobile Service
- Quality-of-Service & Resource Requirements

App 2: Big Data Analytics
- Computational & Storage Requirements

Realization and Embedding

Virtualization and Isolation
Many Algorithmic Challenges

- How to maximize the resource utilization/sharing?
  - E.g., how to embed a maximal number of virtual Hadoop clusters?

- And still ensure a predictable application performance?
  - How to meet the job deadline in MapReduce application?
  - How to guarantee low lookup latencies in data store?
  - It’s not only about resource contention! Skew due to high demand also occurs in well-provisioned systems

- How to exploit allocation flexibilities to even mask and compensate for unpredictable events (e.g., failures)?
  - A key benefit of virtualization!
It’s a Great Time to Be a Scientist

“We are at an interesting inflection point!”

Programmability and virtualization

Algorithms

Confluence: innovation!

Keynote by George Varghese at SIGCOMM 2014
Challenges of More Flexible Distributed Systems

1. **Kraken**: Predictable cloud application performance through adaptive virtual clusters

2. **C3**: Low tail latency in cloud data stores through replica selection

3. **Panopticon**: How to introduce these innovative technologies in the first place? Case study: SDN

4. **STN, Offroad, Peacock**: How to render distributed systems more adaptive without shooting in your foot?
Challenges of More Flexible Distributed Systems

1. **Kraken**: Predictable cloud application performance through adaptive virtual clusters

2. **C3**: Low tail latency in cloud data stores through replica selection

3. **Panopticon**: How to introduce these innovative technologies in the first place? Case study: SDN

4. **STN, Offroad, Peacock**: How to render distributed systems more adaptive without shooting in your foot?
Challenges of More Flexible Distributed Systems

1. **Kraken**: Predictable cloud application performance through adaptive virtual clusters

2. **C3**: Low tail latency in cloud data stores through replica selection

3. **Panopticon**: How to introduce these innovative technologies in the first place? Case study: SDN

4. **STN and Offroad**: How to render distributed systems more adaptive without shooting in your foot?
Cloud Computing + Networking?!

*Network matters!*

- Example: Batch Processing Applications such as Hadoop
  - Communication intensive: e.g., shuffle phase
  - Example Facebook: 33% of execution time due to communication

- For predictable performance in shared cloud: need explicit bandwidth reservations!

- How to max utilization? A network embedding problem!
Let’s Exploit Allocation Flexibilities to Maximize Utilization
Let’s Exploit Allocation Flexibilities to Maximize Utilization

Start simple: exploit flexible routing between given VMs
Let’s Exploit Allocation Flexibilities to Maximize Utilization

Start simple: exploit flexible routing between given VMs

- Integer multi-commodity flow problem with 2 flows?
Let’s Exploit Allocation Flexibilities to Maximize Utilization

Start simple: exploit flexible routing between given VMs
- Integer multi-commodity flow problem with 2 flows?
- Oops: NP-hard
Let’s Exploit Allocation Flexibilities to Maximize Utilization

Start simple: exploit flexible routing between given VMs

- Integer multi-commodity flow problem with 2 flows?
- Oops: NP-hard

Forget about paths: exploit VM placement flexibilities!

- Most simple: Minimum Linear Arrangement without capacities
Let’s Exploit Allocation Flexibilities to Maximize Utilization

Start simple: exploit flexible routing between given VMs

- Integer multi-commodity flow problem with 2 flows?
- Oops: NP-hard

Forget about paths: exploit VM placement flexibilities!

- Most simple: Minimum Linear Arrangement without capacities
- NP-hard 😞
That's all Folks!
Theory vs Practice

Goal in theory:
Embed as general as possible *guest graph* to as general as possible *host graph*

Reality:
Datacenters, WANs, etc. exhibit much *structure* that can be exploited! But also guest networks come with *simple specifications*
Virtual Clusters

- A prominent abstraction for batch-processing applications: Virtual Cluster $VC(n,b)$
- Connects $n$ virtual machines to a «logical» switch with bandwidth guarantees $b$
- A simple abstraction
How to embed a Virtual Cluster in a Fat-Tree?

- Example: dynamic programming

Dynamic Program = optimal solutions for subproblems can efficiently be combined into an optimal solution for the larger problem!
How to embed a Virtual Cluster in a Fat-Tree?

- Example: dynamic programming

Dynamic Program = optimal solutions for subproblems can efficiently be combined into an optimal solution for the larger problem!
How to embed a Virtual Cluster in a Fat-Tree?

Dynamic Program = optimal solutions for subproblems can efficiently be combined into an optimal solution for the larger problem!

How to optimally embed $x$ VMs here, $x \in \{0, \ldots, n\}$?

Cost = 0 or $\infty$!
How to embed a Virtual Cluster in a Fat-Tree?

Dynamic Program = optimal solutions for subproblems can efficiently be combined into an optimal solution for the larger problem!

Cost[x] = min_y Cost[y] + Cost[x-y] + cross-traffic + connections to v

t = 1: solve height 1!
How to embed a Virtual Cluster in a Fat-Tree?

Dynamic Program = optimal solutions for subproblems can efficiently be combined into an optimal solution for the larger problem!

\[ \text{Cost}[x] = \min_y \text{Cost}[y] + \text{Cost}[x-y] + \text{cross-traffic} + \text{connections to } v \]

Or just account on upward link (number of leaving links!)
How to embed a Virtual Cluster in a Fat-Tree?

Dynamic Program = optimal solutions for subproblems can efficiently be combined into an optimal solution for the larger problem!

t = 2: solve height 2!
How to embed a Virtual Cluster in a General Graph?

How to embed?

Guest Graph

Host Graph

1 Unit BW on each link

VM (2Slots)
How to embed a Virtual Cluster in a General Graph?

Algorithm:
- Try all possible locations for virtual switch
- Extend network with artificial source $s$ and sink $t$
- Add capacities
- Compute min-cost max-flow from $s$ to $t$
  (or simply: min-cost flow of volume $n$)
How to embed a Virtual Cluster in a General Graph?

Algorithm:
- Try all possible locations for virtual switch
- Extend network with artificial source \( s \) and sink \( t \)
- Add capacities
- Compute min-cost max-flow from \( s \) to \( t \)
  (or simply: min-cost flow of volume \( n \))
How to embed a Virtual Cluster in a General Graph?

Algorithm:
- Try all possible locations for virtual switch
- Extend network with artificial source s and sink t
- Add capacities
- Compute min-cost max-flow from s to t
  (or simply: min-cost flow of volume n)

capacity = \(\text{floor}(\text{available resources} / \text{unit demand})\)

enough to embed n VMs
How to embed a Virtual Cluster in a General Graph?

Algorithm:
- Try all possible locations for virtual switch
- Extend network with artificial source $s$ and sink $t$
- Add capacities
- Compute min-cost max-flow from $s$ to $t$
  (or simply: min-cost flow of volume $n$)

Guaranteed integer if links are integer!
(E.g., successive shortest paths)
Predictable Performance with Kraken

- This algorithm is used in our system Kraken

- Gives compute and network guarantees... but reality is more complicated:
  - Static resource reservations are inefficient: want to change reservations / virtual clusters!
  - It is also hard to predict resource requirements, stragglers, failures, job executions: want to be online

- Kraken allows to upgrade and downgrade resources in an online fashion, while providing minimal isolation guarantees
The need for adjustments

Constant reservations would be wasteful:

Bandwidth utilization of a TeraSort job over time.

In red: Kraken’s bandwidth reservation.

(Tasks inform Hadoop controller prior to shuffle phase; reservation with Linux tc.)
The need for online adjustments

- *Temporal* resource patterns are hard to predict
- Resource allocations must be changed *online*

>20% variance

>50% variance in killed tasks

Bandwidth utilization of 3 different runs of the same TeraSort workload (without interference)

Completion times of jobs in the presence of speculative execution (left) and the number of speculated tasks (right)
Kraken: Online Reconfigurations

Kraken provides:

- Predictable performance through **bandwidth reservations**
- **Resource-minimal** embeddings
- Support for **online** resource adjustments
- Support for **migration**

Upgrades may require migrations:

---

Fuerst, Schmid, Suresh, Costa
SIGMETRICS 2015
Kraken: Predictable Performance

- Kraken is immune to interference (from iperf):

![Graph showing Map and reduce progress versus time with No reservations, kraken, Map progress, and Reduce progress lines.]

*Kraken (in Hadoop-YARN) with iperf cross-traffic*
There is no infinite lunch: QoS also Requires Admission Control

- Which ones to accept?
- Online primal-dual approach

Even, Medina, Schaffrath, Schmid
TCS 2013
Online Admission Control: General Model

Traffic models

Customer Pipe
Traffic matrix: Bandwidth per VM pair \((u,v)\)

Hose Model
Per VM bandwidth: polytope of traffic matrices.

Aggregate Ingress
Only ingress specified: e.g., support multicast etc.

Routing models

Tree
Steiner tree embedding

Single Path
Unsplittable paths

Multi-Path
Splittable paths (more capacity)

Relay costs: e.g., depending on packet rate
Online Admission Control: Primal-Dual

Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the $j$th round:

1. $f_{j,\ell} \leftarrow \arg\min\{\gamma(j, \ell) : f_{j,\ell} \in \Delta_j\}$ (oracle procedure)
2. If $\gamma(j, \ell) < b_j$ then, (accept)
   (a) $y_{j,\ell} \leftarrow 1$.
   (b) For each row $e : A_{e,(j,\ell)} \neq 0$ do
       $x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/ce} + \frac{1}{w(j, \ell)} \cdot (2^{A_{e,(j,\ell)}/ce} - 1)$.
   (c) $z_j \leftarrow b_j - \gamma(j, \ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0$. 

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.
Online Admission Control: Primal-Dual

Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).
Upon the \( j \)th round:

1. \( f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j, \ell) : f_{j,\ell} \in \Delta_j\} \) (oracle procedure)
2. If \( \gamma(j, \ell) < b_j \) then, (accept)
   a. \( y_{j,\ell} \leftarrow 1 \).
   b. For each row \( e : A_{e,(j,\ell)} \neq 0 \) do
      \[
x_e \leftarrow x_e + \frac{2^{A_{e,(j,\ell)}/c_e}}{w(j, \ell)} \cdot (2^{A_{e,(j,\ell)/c_e}} - 1) - 1.
      \]
   c. \( z_j \leftarrow b_j - \gamma(j, \ell) \).
3. Else, (reject)
   a. \( z_j \leftarrow 0 \).

Formulate the packing (dual) LP: Maximize profit
(Note: dynamic LP!)

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Does not know \( t'>t \).
Competitive ratio:
\[
r = \frac{\text{Cost(ON)}}{\text{Cost(OFF)}}
\]
**Algorithm 1** The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the $j$th round:

1. $f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j, \ell) : f_{j,\ell} \in \Delta_j\}$ (oracle procedure)
2. If $\gamma(j, \ell) < b_j$ then, (accept)
   (a) $y_{j,\ell} \leftarrow 1$.
   (b) For each row $e : A_{e,(j,\ell)} \neq 0$ do
      $$x_e \leftarrow x_e \cdot \frac{2^{A_{e,(j,\ell)} / c_e}}{w(j, \ell)} \cdot \left(\frac{2^{A_{e,(j,\ell)} / c_e}}{w(j, \ell)} - 1\right).$$
   (c) $z_j \leftarrow b_j - \gamma(j, \ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0.$
Online Admission Control: Primal-Dual

Algorithm

Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the $j$th round:

1. $f_{j,\ell} \leftarrow \text{argmin}\{\gamma(j, \ell) : f_{j,\ell} \in \Delta_j\}$ (oracle procedure)
2. If $\gamma(j, \ell) < b_j$ then, (accept)
   (a) $y_{j,\ell} \leftarrow 1.$
   (b) For each row $e : A_{e,(j,\ell)} \neq 0$ do
      \[ x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j, \ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1). \]
   (c) $z_j \leftarrow b_j - \gamma(j, \ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0.$

Competitive Analysis

Does not know $t' > t$.
Competitive ratio:

$$r = \frac{\text{Cost(ON)}}{\text{Cost(OFF)}}$$
Primal and Dual

Online Admission Control: Primal-Dual

\[
\begin{align*}
\text{(I)} & \quad \min Z_j^T \cdot 1 + X^T \cdot C \\
& \quad \text{s.t.} \quad Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T \\
& \quad \quad \quad \quad \quad \quad X, Z_j \geq 0
\end{align*}
\]

\[
\begin{align*}
\text{(II)} & \quad \max B_j^T \cdot Y_j \\
& \quad \text{s.t.} \quad A_j \cdot Y_j \leq C \\
& \quad \quad \quad \quad \quad \quad D_j \cdot Y_j \leq 1 \\
& \quad \quad \quad \quad \quad \quad Y_j \geq 0
\end{align*}
\]

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

**Algorithm 1** The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the \(j\)th round:

1. \(f_{j,\ell} \leftarrow \arg\min \{\gamma(j, \ell) : f_{j,\ell} \in \Delta_j\}\) (oracle procedure)
2. If \(\gamma(j, \ell) < b_j\) then, (accept)
   (a) \(y_{j, \ell} \leftarrow 1\).
   (b) For each row \(e : A_{e, (j, \ell)} \neq 0\) do
       \[
x_e \leftarrow x_e \cdot 2^{A_{e, (j, \ell)} / c_e} + \frac{1}{w(j, \ell)} \cdot (2^{A_{e, (j, \ell)} / c_e} - 1).
\]
   (c) \(z_j \leftarrow b_j - \gamma(j, \ell)\).
3. Else, (reject)
   (a) \(z_j \leftarrow 0\).
Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the jth round:

1. \( f_{j,\ell} \leftarrow \arg \min \{ \gamma(j, \ell) : f_{j,\ell} \subset A_j \} \) (oracle procedure)
2. If \( \gamma(j, \ell) < b_j \) then, (accept)
   (a) \( y_{j,\ell} \leftarrow 1 \).
   (b) For each row \( e \) : If \( A_{e,(j,\ell)} \neq 0 \) do
       \[ x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j, \ell)} \cdot (\frac{2^{A_{e,(j,\ell)}/c_e} - 1}{w(j, \ell)}) \]
   (c) \( z_j \leftarrow b_j - \gamma(j, \ell) \).
3. Else, (reject)
   (a) \( z_j \leftarrow 0 \).
Competitive Analysis

Does not know $t'>t$.
Competitive ratio:
\[ r = \frac{\text{Cost(ON)}}{\text{Cost(OFF)}} \]

Primal and Dual

Online Admission Control: Primal-Dual

\[
\begin{align*}
\text{(I)} & \quad \min Z_j^T \cdot 1 + X^T \cdot C \quad \text{s.t.} \\
& \quad Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T \\
& \quad X, Z_j \geq 0
\\
\text{(II)} & \quad \max B_j^T \cdot Y_j \quad \text{s.t.} \\
& \quad A_j \cdot Y_j \leq C \\
& \quad D_j \cdot Y_j \leq 1 \\
& \quad Y_j \geq 0
\end{align*}
\]

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

Algorithm

**Algorithm 1** The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the $j$th round:

1. $f_{j,\ell} \leftarrow \arg\min \{ \gamma(j, \ell) : f_{j,\ell} \in \Delta_j \}$ (oracle procedure)
2. If $\gamma(j, \ell) < b_j$ then, (accept)
   (a) $y_{j,\ell} \leftarrow 1$.
   (b) For each row $e$ : If $A_{e,(j,\ell)} \neq 0$ do
      \[ x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j,\ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1). \]
      (c) $z_j \leftarrow b_j - \gamma(j, \ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0$.

If cheap: accept and update primal variables (always feasible solution)
Algorithm 1 The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the jth round:

1. \( f_{j,\ell} \leftarrow \arg\min\{\gamma(j, \ell) : f_{j,\ell} \in \Delta_j\} \) (oracle procedure)
2. If \( \gamma(j, \ell) < b_j \) then, (accept)
   a. \( y_{j,\ell} \leftarrow 1 \).
   b. For each row \( e : A_{e,(j,\ell)} \neq 0 \) do
      \[ x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j, \ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1). \]
3. Else, (reject)
   a. \( z_j \leftarrow 0 \).

Else reject
Primal and Dual Online Admission Control: Primal-Dual

**Algorithm 1** The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the $j$th round:

1. $f_{j, \ell} \leftarrow \text{argmin}\{\gamma(j, \ell) : f_{j, \ell} \in \Delta_j\}$ \hspace{1cm} (oracle procedure)
2. If $\gamma(j, \ell) < b_j$ then, (accept)
   (a) $y_{j, \ell} \leftarrow 1$.
   (b) For each row $e$ : If $A_{e,(j,\ell)} \neq 0$ do
      
      \[ x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j, \ell)} \cdot (2^{A_{e,(j,\ell)/c_e} - 1). \]
   (c) $z_j \leftarrow b_j - \gamma(j, \ell)$.
3. Else, (reject)
   (a) $z_j \leftarrow 0.$

**Competitive Analysis**

Does not know $t' > t$.
Competitive ratio: $r = \text{Cost(ON)}/\text{Cost(OFF)}$
Primal and Dual  

Online Admission Control: Primal-Dual

\[
\begin{align*}
\text{min } & Z_j^T \cdot 1 + X^T \cdot C \\ s.t. & Z_j^T \cdot D_j + X^T \cdot A_j \geq B_j^T \\
& X, Z_j \geq 0
\end{align*}
\]

\[
\begin{align*}
\text{max } & B_j^T \cdot Y_j \\ s.t. & A_j \cdot Y_j \leq C \\
& D_j \cdot Y_j \leq 1 \\
& Y_j \geq 0
\end{align*}
\]

Fig. 1: (I) The primal covering LP. (II) The dual packing LP.

**Algorithm**

**Algorithm 1** The General Integral (all-or-nothing) Packing Online Algorithm (GIPO).

Upon the \(j\)th round:

1. \(f_{j,\ell} \leftarrow \arg\min \{ \gamma(j, \ell) : f_{j,\ell} \in \Delta_j \}\) \text{(oracle procedure)}
2. If \(\gamma(j, \ell) < b_j\) then, (accept)
   (a) \(y_{j,\ell} \leftarrow 1\).
   (b) For each row \(e\): If \(A_{e,(j,\ell)} \neq 0\) do
      \[x_e \leftarrow x_e \cdot 2^{A_{e,(j,\ell)}/c_e} + \frac{1}{w(j, \ell)} \cdot (2^{A_{e,(j,\ell)}/c_e} - 1).\]
   (c) \(z_j \leftarrow b_j - \gamma(j, \ell)\).
3. Else, (reject)
   (a) \(z_j \leftarrow 0\).

Computationally hard! Use your favorite approximation algorithm! If competitive ratio \(\rho\) and approximation \(r\), overall competitive ratio \(\rho \ast r\).
Challenges of More Flexible Distributed Systems

1. **Kraken**: Predictable cloud application performance through adaptive virtual clusters

2. **C3**: Low tail latency in cloud data stores through replica selection

3. **Panopticon**: How to introduce these innovative technologies in the first place? Case study: SDN

4. **STN, Offroad, Peacock**: How to render distributed systems more adaptive without shooting in your foot?
Latency-Critical Applications

- Another critical requirement besides bandwidth, especially in cloud data stores is **latency**
- Today’s interactive **web** applications require **fluid** response time
- Degraded user experience directly impacts **revenue**

- **Tail** matters...
  - Web applications = multi-tier, **large** distributed systems
  - 1 request involves **10(0)s** data accesses / servers!
  - A **single late** read may delay entire request
How to cut tail latency?

- How to guarantee low tail in shared cloud? A non-trivial challenge even in **well-provisioned** systems
  - **Skews** in demand, time-varying service times, stragglers, ...
  - No time to make make rigorous optimizations or reservations

- **Idea C3**: Exploit **replica selection**!
  - Many distributed DBs resp. **key-value stores** have redundancy
  - **Opportunity** often overlooked so far

- **Our focus**: **Cassandra** (1-hop DHT, server = client)
  - Powers, e.g., Ebay, Netflix, Spotify
  - More sophisticated than MongoDB or Riak
C3: Exploit Replica Selection

- Great idea! But how? Just go for «the best»?
Careful: «The best» can change

- Not so simple!
  - Need to deal with **heterogenous** and **time-varying** service times
  - Background garbage collection, log compaction, TCP, deamons

{request} →

- ![Diagram showing a request being sent to three servers with different service times](image-url)
  - 4 ms
  - 5 ms
  - 80 ms
Careful: Herd Behavior

- Potentially high fan-in and herd behavior!
- Observed in Cassandra Dynamic Snitching (DS)
  - Coarse time intervals and I/O gossiping
  - Synchronization and stale information

A coordination / control theory problem!
4 Principles:

- Stay informed: **piggy-back** queue state and service times
- Stay reactive and don’t commit: use **backpressure queue**
- Leverage heterogeneity: **compensate** for service times
- Avoid redundancy

Mechanism 1: replica ranking

- Penalize larger queues

Mechanism 2: rate control

- Goal: match service rate and keep pipeline full
- Cubic, with saddle region

Suresh, Canini, Schmid, Feldmann NSDI 2015
Performance Evaluation

- Methodology:
  - Amazon EC2
  - disk vs SSD
  - BigFoot testbed
  - Simulations

- Lower tail latency
  - 2-3x for 99.9%

- Higher read throughput...

- ... and lower load (and variance)!
Challenges of More Flexible Distributed Systems

1. **Kraken**: Predictable cloud application performance through adaptive virtual clusters

2. **C3**: Low tail latency in cloud data stores through replica selection

3. **Panopticon**: How to introduce these innovative technologies in the first place? Case study: SDN

4. **STN, Offroad, Peacock**: How to render distributed systems more adaptive without shooting in your foot?
SDN Use Cases Today

Many use cases discussed today, e.g. in:
- Enterprise networks
- Datacenters
- WANs
- IXPs
- ISPs

How to deploy SDN cost effectively?
SDN Deployment

Datacenter: Easy

- SDN can be deployed at **software edge** (terminate links at Open vSwitch)
- 2 Control Planes: **ECMP Fabric**

WAN: «Easy»

- Google B4: **small network**
- Can be deployed at end of long-haul fiber (replace IP core router)
SDN Deployment

Datacenter: Easy
- SDN can be deployed at software edge (terminate links at Open vSwitch)
- 2 Control Planes: ECMP Fabric

WAN: «Easy»
- Google B4: small network
- Can be deployed at end of long-haul fiber (replace IP core router)

Problem: first benefits only at “flag day” (only control plane incremental)
But how to deploy SDN in enterprise?

- Large and complex networks, budgets limited

- Idea: Can we incrementally deploy SDN into enterprise campus networks?

- And what SDN benefits can be realized in a hybrid deployment?
Can we deploy SDN at enterprise edge?

The edge is large, and not in software!
TOOL
Determine the partial SDN deployment

SDN ARCHITECTURE
Operate the network as a (nearly) full SDN
Get Functionality with Waypoint Enforcement

Insight #1: 
≥ 1 SDN switch → Policy enforcement

Middlebox traversal

Access control
Larger Deployment = More Flexibility

Insight #1: ≥ 1 SDN switch → Policy enforcement

Insight #2: ≥ 2 SDN switches → Fine-grained control

Traffic load-balancing
Panopticon: Building the Logical SDN Abstraction

1. Restrict traffic by using VLANs
Panopticon: Building the Logical SDN Abstraction

2. Build logical SDN

“Logical SDN”
PANOPTICON provides the abstraction of a (nearly) fully-deployed SDN in a partially upgraded network
Good or Bad Impact on Traffic?

1. Congestion

2. Harvest unutilized network capacity
Challenges of More Flexible Distributed Systems

1. **Kraken**: Predictable cloud application performance through adaptive virtual clusters

2. **C3**: Low tail latency in cloud data stores through replica selection

3. **Panopticon**: How to introduce these innovative technologies in the first place? Case study: SDN

4. **STN, Offroad, Peacock**: How to render distributed systems more adaptive without shooting in your foot?
Correct Operation is Important!

Example: trend to move the infrastructure to the cloud (e.g., the CIA).

What if your traffic was *not* isolated from other tenants during periods of routine *maintenance*?
Example: Outages

Even technically sophisticated companies are struggling to build networks that provide reliable performance.

We discovered a misconfiguration on this pair of switches that caused what's called a “bridge loop” in the network.

A network change was [...] executed incorrectly [...] more “stuck” volumes and added more requests to the re-mirroring storm

Service outage was due to a series of internal network events that corrupted router data tables

Experienced a network connectivity issue [...] interrupted the airline's flight departures, airport processing and reservations systems
Example: Security-Critical Updates

attacker

old route r1

security critical area
Example: Security-Critical Updates

attacker

old route r1

Waypoint Enforcement

security critical area
Example: Security-Critical Updates

- **Attacker**
- **Waypoint Enforcement**
- **Old Route r1**
- **New Route r2**
- **Security Critical Area**

Diagram shows an attacker attempting to access a security critical area via an old route r1. A new route r2 is implemented to bypass the security critical area.
Example: Security-Critical Updates

Controller Updates

security critical area

new route r2

old route r1

Waypoint Enforcement

Example: Security-Critical Updates
How to update networks consistently?

- Idea: Use tagging and 2-phase commit
  - Problematic: header space, TCAM space, middleboxes

  - Round = subset of nodes are updated
  - Restrict concurrency s.t. consistency maintained
  - How many rounds are needed?

Solution: Round 1

Controller Updates

attacker

old route r1

new route r2

Waypoint Enforcement

security critical area

$
Solution: Round 2

Controller Updates

attacker

Waypoint Enforcement

old route r1

new route r2

security critical area

Controller Updates
Solution

- How many rounds are needed?
- How to also avoid loops? Related to \textbf{Feedback Arc Set Problems}
- What properties conflict?
- \textbf{NP-hard} but efficient algorithms exist!

Ludwig, Rost, Fourcard, Schmid
HotNets 2014
Ludwig, Marcinkowski, Schmid
PODC 2015
Distributed Control: for redundancy, multi-user, ...

Control should be distributed!

**STN**: A transactional interface

**Problem**: Conflict free, per-packet consistent policy composition and installation

**Holy Grails**: Linearizability (Safety), Wait-freedom (Liveness)

*Canini, Kuznetsov, Levin Schmid*  
INFOCOM 2015
Challenge: Fast Robust Routing Mechanisms

- **Link failures** today are not uncommon [1]

- Modern networks provide **robust routing mechanisms**
  - i.e., routing which reacts to failures
  - example: MPLS local and global path protection

Before failover:

After failover:
Fast In-band Failover

• Important that failover happens fast = in-band
  • Reaction time in control plane can be orders of magnitude slower

• For this reason: **OpenFlow Local Fast Failover Mechanism**
  • Supports conditional forwarding rules (depend on the local state of the link: live or not?)

• Gives fast but local and perhaps “suboptimal” forwarding sets
  • Controller improves globally later…
Important that failover happens fast = in-band
  • Reaction time in control plane can be orders of magnitude slower

For this reason: OpenFlow Fast Failover Mechanism
  • Supports conditional forwarding rules (depend on the local state of the link: live or not?)

Gives fast but local and perhaps “suboptimal” forwarding sets
  • Controller improves globally later...

However, not much is known about how to use the OpenFlow fast failover mechanism. E.g.: How many failures can be tolerated without losing connectivity?
• Important that failover happens fast = in-band
  • Reaction time in control plane can be orders of magnitude slower

• For this reason: OpenFlow Local Fast Failover Mechanism
  • Supports conditional forwarding rules (depend on the local state of the link: live or not?)
  • Gives fast but local and perhaps "suboptimal" forwarding sets
  • Controller improves globally later…

However, not much is known about how to use the OpenFlow fast failover mechanism. E.g.: How many failures can be tolerated without losing connectivity?

How to use mechanism is a non-trivial problem even if underlying network stays connected: (1) conditional failover rules need to be allocated ahead of time, without knowing actual failures, (2) views at runtime are inherently local. How not to shoot in your foot with local fast failover (e.g., create forwarding loops)?
Offroad and SmartSouth

- **Offroad**: already with today’s Openflow, provable connectivity can be implemented in-band
  - Even without per-switch state

- **SmartSouth**: already with today’s Openflow, many additional functionality could in principle be implemented in-band
  - E.g., anycast, sampling, snapshots, blackhole detection, ...

- Trend for «Openflow 2.0»: improve functionality of Openflow switches further
  - Registers, bitmasking, no longer field-specific, ...

Schiff, Borokhovich, Schmid
HotSDN 2014, HotNets 2014
Conclusion

- Programmable and virtualized systems: **opportunities** for improved resource allocation and utilization

- But also **challenges** in terms of resource interference and predictable application performance

- Making the network a **first class citizen** can help to improve performance

- **High potential** but also **risks** of a more dynamic control

Thank you!

And thanks to my co-authors, mainly: Marco Canini, Paolo Costa, Carlo Fürst, Petr Kuznetsov, Dan Levin, Arne Ludwig, Matthias Rost, Jukka Suomela, Lalith Suresh
References


Backup Slides
Flavors of VNet Embedding Problems (VNEP)

Minimize embedding footprint of a single VNet:

Maximize profit over time:

Minimize max load of multiple VNets or collocate to save energy:

Endpoints fixed:

Time
Flavors of VNet Embedding Problems (VNEP)

**Great opportunities?:** Already for a line host graph, computing the footprint and load optimal embedding of a single VNet is NP-hard (e.g., minimum linear arrangement).

**And:** Generalization of Online Call Control for entire networks, plus embedding problem on top!
Cannot directly apply minor theory!

It is possible to embed a guest graph $G$ on a host graph $H$, even though $G$ is not a minor of $H$:

Planar Graph $H$: $K_5$ and $K_{3,3}$ minor-free...

... but possible to embed $G=K_5$!
Online Access Control (1)

- Assume: end-point locations given
- Different routing and traffic models
- Price and duration
- Which ones to accept?
- Online Primal-Dual Framework (Buchbinder and Naor)
Solving the VNEP

- Formulate a Mixed Integer Program!
- Leverage additional structure!
- Use online primal-dual approach

**Discussion:**
- Virtual network embedding a potential threat?
- Adding migration support
- Beyond graph structures
Security Aspects

Find dense parts first! But careful:
A cannot be embedded in B.
B cannot be embedded in A.
But A can be embedded in B.

Different from minor relation:
Can embed cliques in planar graphs.

MinCut? Topology?

Algorithm

Knitting  Expand links  Repeat
Migration

- Service or CloudNet migration
- Access cost: latency
- Migration cost: service interruption / bandwidth
- Variant of Uniform Metrical Task System (graph-based access)
- Allows for $O(\log n / \log \log n)$ solutions (unlike MTS)

Amortized migration:

Lower bound: Online function tracking
Migration: Example

- Single service
- Migration Cost $m$
- Access Cost $1$
- Goal: minimize sum of both?

Realm of competitive analysis!
Migration: Example

- $O(\log n)$ competitive ratio only
- $O(\log n / \log\log n)$ not elegant (yet)
Migration: Example

- O(log n) competitive ratio only
- O(log n / loglog n) not elegant (yet)

Deterministic Algo: Amortize!
1. Access cost counters at each node (if service there)
2. When counter exceeds m, deactivate nodes with counters > m/2, migrate to active node in center of active component: minimal sum of distances
3. When no node left, epoch ends. Reset and restart.
Migration: Example

- O(log n) competitive ratio only
- O(log n / loglog n) not elegant (yet)

@ t = 0:

Deterministic Algo: Amortize!
1. Access cost counters at each node (if service there)
2. When counter exceeds $m$, deactivate nodes with counters $> m/2$, migrate to active node in center of active component: minimal sum of distances
3. When no node left, epoch ends. Reset and restart.
Migration: Example

- O(log n) competitive ratio only
- O(log n / loglog n) not elegant (yet)

@ t = 1:

Deterministic Algo: Amortize!

1. Access cost counters at each node (if service there)
2. When counter exceeds $m$, deactivate nodes with counters > $m/2$, migrate to active node in center of active component: minimal sum of distances
3. When no node left, epoch ends. Reset and restart.
Migration: Example

- \(O(\log n)\) competitive ratio only
- \(O(\log n / \log \log n)\) not elegant (yet)

@ \(t = 1\):

on service!

Deterministic Algo: Amortize!

1. Access cost counters at each node (if service there)
2. When counter exceeds \(m\), deactivate nodes with counters > \(m/2\), migrate to active node in center of active component: minimal sum of distances
3. When no node left, epoch ends. Reset and restart.
Migration: Example

- O(log n) competitive ratio only
- O(log n / loglog n) not elegant (yet)

Deterministic Algo: Amortize!
1. Access cost counters at each node (if service there)
2. When counter exceeds \( m \), deactivate nodes with counters > \( m/2 \), migrate to active node in center of active component: minimal sum of distances
3. When no node left, epoch ends. Reset and restart.

@ \( t = 2 \): on service!
Migration: Example

- O(log n) competitive ratio only
- O(log n / loglog n) not elegant (yet)

Deterministic Algo: Amortize!
1. Access cost counters at each node (if service there)
2. When counter exceeds $m$, deactivate nodes with counters > $m/2$, migrate to active node in center of active component: minimal sum of distances
3. When no node left, epoch ends. Reset and restart.

@ t = 2:
Migration: Example

- O(log n) competitive ratio only
- O(log n / loglog n) not elegant (yet)

@ t = 3: epoch ends!

Deterministic Algo: Amortize!
1. Access cost counters at each node (if service there)
2. When counter exceeds m, deactivate nodes with counters > m/2, migrate to active node in center of active component: minimal sum of distances
3. When no node left, epoch ends. Reset and restart.
Migration: Example

- $O(\log n)$ competitive ratio only
- $O(\log n / \log \log n)$ not elegant (yet)

Deterministic Algo: Amortize!
1. Access cost counters at each node (if service there)
2. When counter exceeds $m$, deactivate nodes with counters > $m/2$, migrate to active node in center of active component: minimal sum of distances
3. When no node left, epoch ends. Reset and restart.

Analysis

Offline algorithm OFF has cost $>m/2$ per epoch:
1. True if OFF migrates at least once.
2. If OFF does not migrate: any single location has access cost $>m/2$.

Online algorithm ON has cost at most $O(m \log n)$ per epoch:
1. Access costs \textit{per phase} at most $m$: counters
2. Migration cost per phase: $m$
3. How many phases? Due to center strategy, at least 1/8-th of active nodes become passive
Solving the VNEP

- Formulate a Mixed Integer Program!

- Leverage additional structure!

- Use online primal-dual approach

Discussion:
- Virtual network embedding a potential threat?
- Adding migration support
- Beyond graph structures
Beyond Graph Specifications

- Example: Multicast with in-network processing
- The topology becomes subject to optimization as well
- Example: Cost efficient multicast or aggregation

Best of both worlds?
Joint optimization!

n unicasts
(43 edges, 0 nodes)

Multicast / Steiner tree
(16 edges, 9 nodes)
Beyond Graph Specifications

- Example: Multicast with in-network processing
- The topology becomes subject to optimization as well
- Example: Cost efficient multicast or aggregation

Substrate:

- n unicasts
  (43 edges, 0 nodes)
- Joint optimization: Virtual Steiner Arborescence
  (26 edges, 2 nodes)
- Multicast / Steiner tree
  (16 edges, 9 nodes)
Beyond Graph Specifications

- Approach: Single-commodity MIP and path decomposition
  - Multi-commodity: 1,200,000 integer variables
  - Single-commodity: 6,000 integer variables
  - But lose information
“(Network) Virtualization: The Killer Application for SDN” (Nick McKeown)

The Internet has changed radically over the last decades

**Historic goal:** Connectivity between a small set of super-computers

**Applications:** File transfer and emails among scientists

**Situation now:** Non-negligible fraction of the world population is constantly online

**New requirements:**

- More traffic, new demands on reliability and predictability
- Thus: use infrastructure more efficiently, use in-network caches: **TE beyond destination-based routing**, …
- Many different applications: Google docs vs datacenter synchronization vs on-demand video
- SDN allows us to schedule and route different applications according to their needs
Rigorous Solutions for the Geneal Embedding Problem: MIP

Recipe:
- A (linear) objective function (e.g., load or footprint)
- A set of (linear) constraints
- Feed it to your favorite solver (CPLEX, Gurobi, etc.)

Details:
- Introduce binary variables $\text{map}(v,s)$ to map virtual nodes $v$ on substrate node $s$
- Introduce flow variables for paths (splittable or not?)
- Ensure flow conservation: all flow entering a node must leave the node, unless it is the source or the destination
Rigorous Solutions for the Geneal Embedding Problem: MIP

**Constants:**
- Substrate Vertices: \( V_s \)
- Substrate Edges: \( E_s : V_s \times V_s \)
- Unique: \( uni_check_s : \forall (s_1, s_2) \in E_s : (s_2, s_1) \notin E_s \)
- SNode Capacity: \( snc(s) \rightarrow \mathbb{R}^+, s \in V_s \)
- SLink Capacity: \( slc(e_s) \rightarrow \mathbb{R}^+, e_s \in E_s \)
- Requests: \( R \)
- Virtual Vertices: \( V_v(r), r \in R \)
- Virtual Edges: \( E_v(r) : \rightarrow V_v(r) \times V_v(r), r \in R \)
- Unique: \( uni_check_v : \forall r \in R, (v_1, v_2) \in E_v(r) : (v_2, v_1) \notin E_v(r) \)
- VNode Demand: \( vnd(r, v) \rightarrow \mathbb{R}^+, r \in R, v \in V_v(r) \)
- VEdge Demand: \( vld(r, e_v) \rightarrow \mathbb{R}^+, r \in R, e_v \in E_v(r) \)

**Edges-Reverse:** \( ER_s : \forall (s_1, s_2) \in E_s \exists (s_2, s_1) \in ER_s \wedge |E_s| = |ER_s| \)

**Migration Cost:** \( mig_cost(r, v, s) \rightarrow \mathbb{R}^+ |V_v(r)| \times |V_s|, r \in R, v \in V_v(r), s \in V_s \)

**Possible Placements:** \( place(r, v, s) \rightarrow \{0, 1\} |V_v(r)| \times |V_s|, r \in R, v \in V_v(r), s \in V_s \)

**Variables:**
- Node Mapping: \( n_map(r, v, s) \in \{0, 1\}, r \in R, v \in V_v(r), s \in V_s \)
- Flow Allocation: \( f_alloc(r, e, eb) \geq 0, r \in R, e \in E_v(r), eb \in EB_s \)

**Constraints:**
- Each Node Mapped: \( \forall r \in R, v \in V_v(r) : \sum_{s \in V_s} n_map(r, v, s) \cdot place(r, v, s) = 1 \)
- Feasible: \( \forall s \in V_s : \sum_{r \in R, v \in V_v(r)} n_map(r, v, s) \cdot vnd(r, v) \leq snc(s) \)
- Guarantee Link Realization: \( \forall r \in R, (v_1, v_2) \in E_v(r), s \in V_s \sum_{(s_1, s_2) \in V_s \times V_s \cap EB_s} f_alloc(r, v_1, v_2, s, s_2) - \sum_{(s_1, s_2) \in V_s \times V_s \cap EB_s} f_alloc(r, v_1, v_2, s_1, s) = vld(r, v_1, v_2) \cdot (n_map(r, v_1, s) - n_map(r, v_2, s)) \)
- Realize Flows: \( \forall (s_1, s_2) \in E_s \sum_{r \in R, (v_1, v_2)} f_alloc(r, v_1, v_2, s_1, s_2) + f_alloc(r, v_1, v_2, s_2, s_1) \leq slc(s_1, s_2) \)

**Objective function:**
- Minimize Embedding Cost: \( \min : \sum_{r \in R, (v_1, v_2) \in E_v(r), (s_1, s_2) \in E_s} f_alloc(r, v_1, v_2, s_1, s_2) + f_alloc(r, v_1, v_2, s_2, s_1) \)

entering a node must leave the node, unless it is the source or the destination
Mixed Integer Programs (1)

- MIPs can be quite fast
  - For pure integer programs, SAT solvers likely faster
- However, that’s not the end of the story: \textit{MIP} \neq \textit{MIP}
  - The specific formulation matters!
- For example: many solvers use relaxations
  - Make integer variables \textit{continuous}: resulting linear programs (LPs) can be solved \textit{in polynomial time}!
  - How good can solution in this subtree (given fixed variables) be at most? (More flexibility: solution can only be better!)
  - If already this is worse than currently best solution, we can \textit{cut}!

- Relaxations can also be used as a basis for heuristics
  - E.g., round fractional solutions to closest integer?
Mixed Integer Programs (2)

Branch & bound tree:

Relax: possible to obtain better solution than we already have?

best so far

best
Mixed Integer Programs (3)

- Recall: Relaxations useful if they give good bounds
- However it’s hard to formulate a MIP for VNEP which yields useful relaxations!
- What happens here?

VNet: Physical Network:
Mixed Integer Programs (3)

- Recall: Relaxations useful if they give good bounds
- However it’s hard to formulate a MIP for VNEP which yields useful relaxations!
- What happens here?

VNet:  

Physical Network:

map(v,s)=.5
Mixed Integer Programs (3)

- Recall: Relaxations useful if they give good bounds
- However it’s hard to formulate a MIP for VNEP which yields useful relaxations!
- What happens here?

VNet:  

Physical Network:

- Flow = 0
Mixed Integer Programs (3)

- Recall: Relaxations useful if they give good bounds
- However it’s hard to formulate a MIP for VNEP which yields useful relaxations!
- What happens here?

VNet: Physical Network:

Relaxations do not provide good bounds: allocation 0! Also not useful for rounding...
Example 1: Embedding

Where to allocate my virtual machines?

- For a **predictable performance**, try to avoid interference! Keep it **local**!
- Or make explicit **bandwidth reservations**! And keep it local to keep reservations small.
- .... but avoid static bandwidth reservations and make resource reservations **in online fashion**.