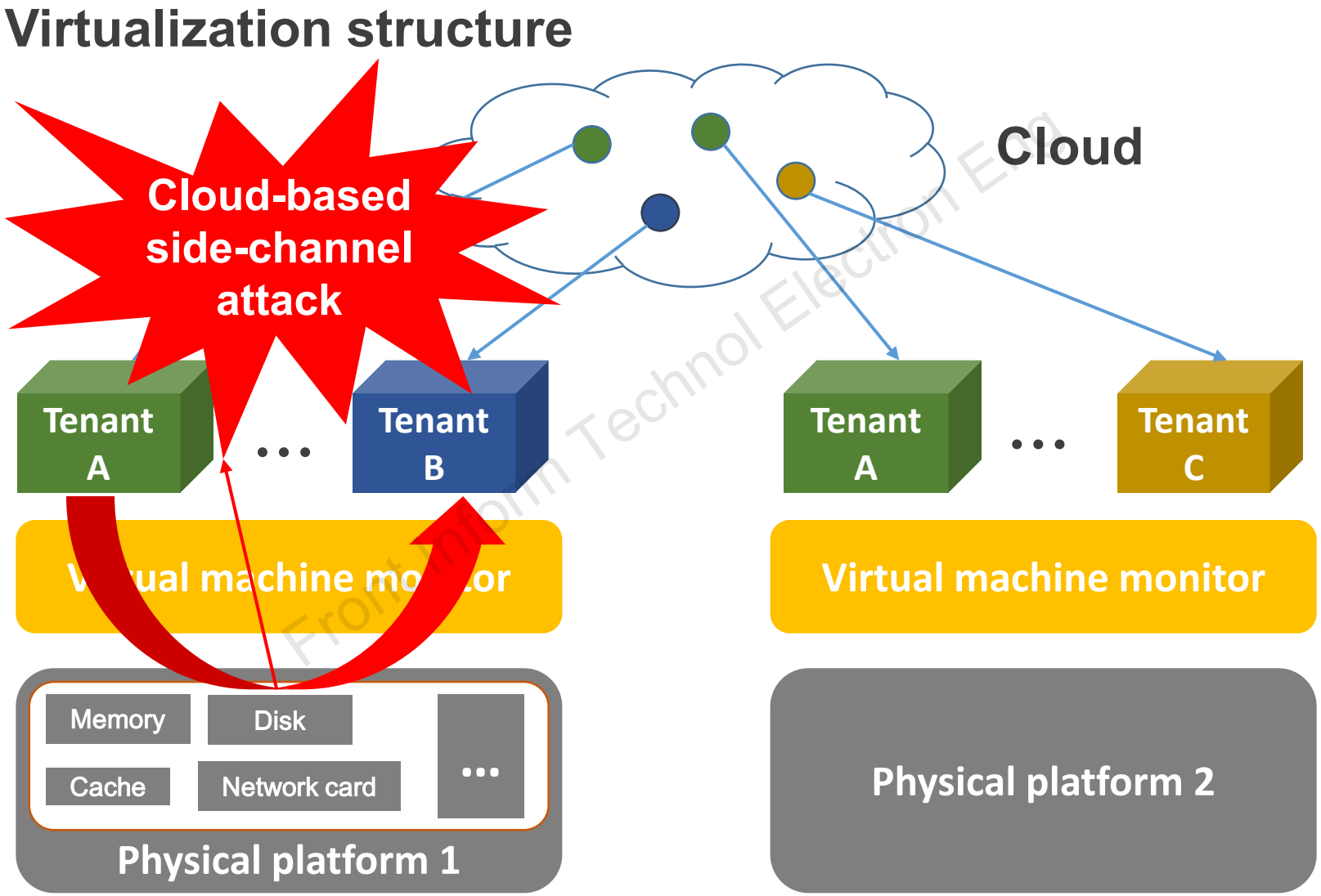
Chao YANG, Yun-fei GUO, Hong-chao HU, Ya-wen WANG, Qing TONG, Ling-shu LI, 2019. Driftor: mitigating cloud-based side-channel attacks by switching and migrating multi-executor virtual machines. *Frontiers of Information Technology & Electronic Engineering*, 20(5):731-748. https://doi.org/10.1631/FITEE.1800526

Driftor: mitigating cloud-based side-channel attacks by switching and migrating multi-executor virtual machines

Key words: Cloud computing; Side-channel attack; Information leakage; Multi-executor structure; Virtual machine switch; Virtual machine migration

Corresponding author: Chao YANG
E-mail: 1989600235@qq.com
ORCID: http://orcid.org/0000-0002-4796-7011

Motivation (1/2)



Motivation (2/2)

VM migration-based defense

Weaknesses:

1. No unified and practical adversary model for cloud-based side channels

2. Unable to deal with fast side-channel attacks

Strengths:

1. Inherit mechanism in all cloud systems

2. Dealing with a variety of side-channel attacks due to co-residency of tenants

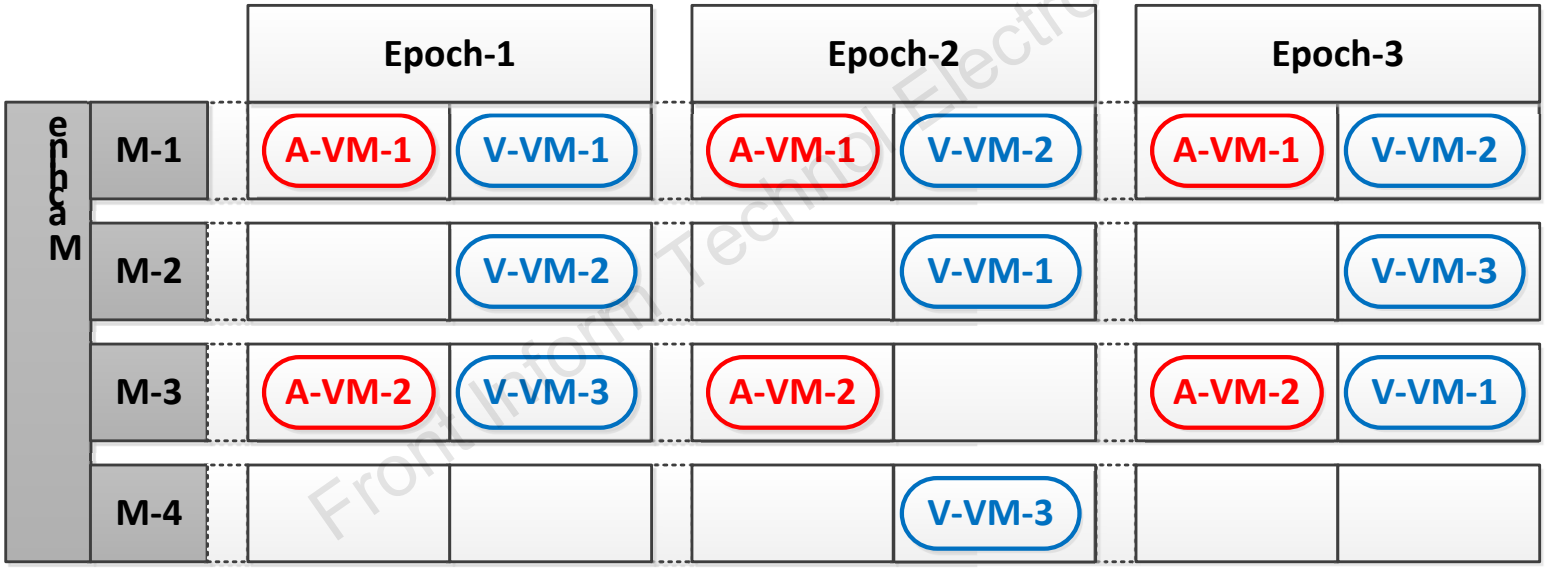
Main idea

Imitative “VM migration” to improve defense ability

- Set up a practical and unified adversary model where the attacker focuses on effective side-channel attacks.
- Design a multi-executor structure for virtual machines in cloud, where there is only one active executor to provide service.
- Propose a satisfying scheme for executor switching active state between executors and migrating VMs between servers.
- Evaluate the security improvement and the performance of Driftor with simulation and in a local real-world cloud environment.

Method (1/9)

Adversary model—information leakage between tenants due to their co-residency



Co-residency of tenant A and tenant V in three epochs

Method (2/9)

Adversary model—information leakage between tenants due to their co-residency

Three factors affecting the amount of information leakage

	V-VM-1	V-VM-2	V-VM-3
A-VM-1	1	2	0
A-VM-2	1	0	1

Factor 1: across time

	Victim
A-VM-1	3
A-VM-2	2

Factor 2: information replication of the victim's VMs

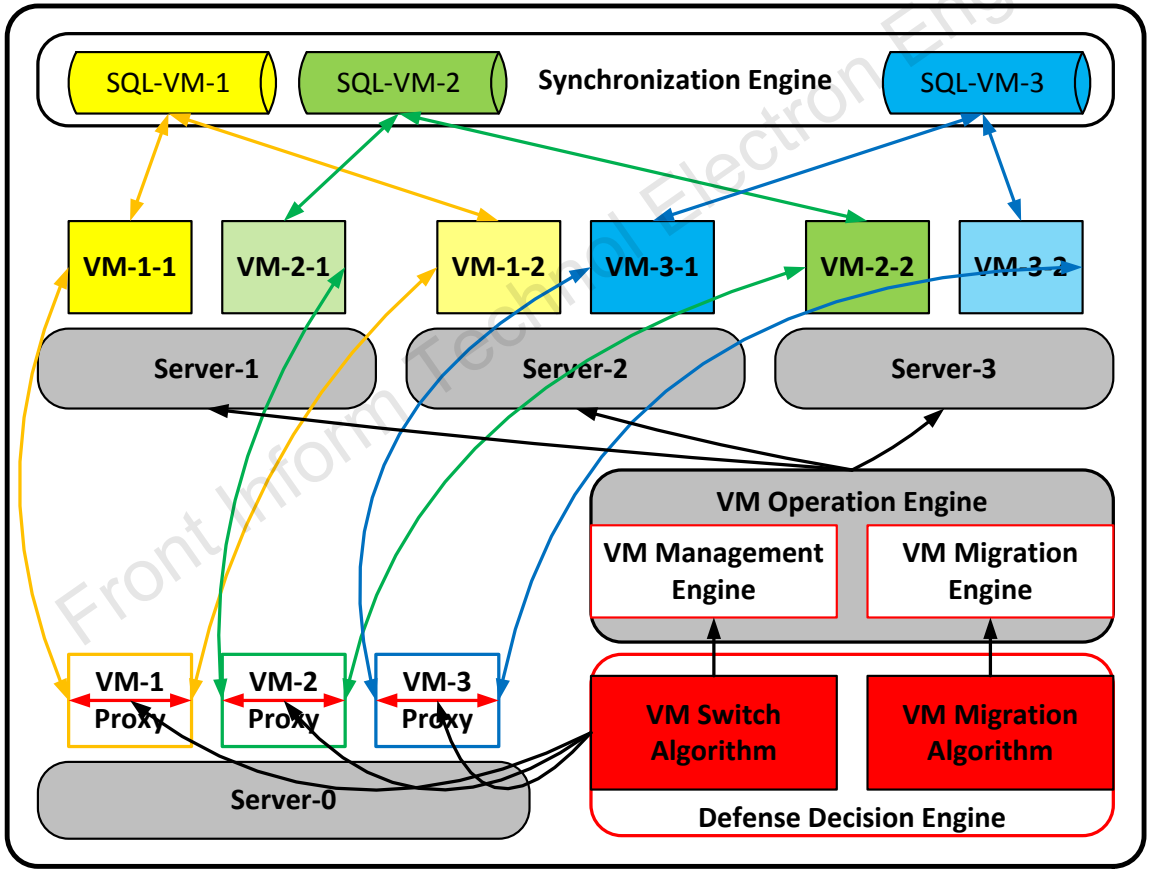
	Victim
Adversary	5

Factor 3: information sharing across the attacker's VMs

Effective side-channel attacks:
If enough information is extracted during five epochs, this attack is effective.

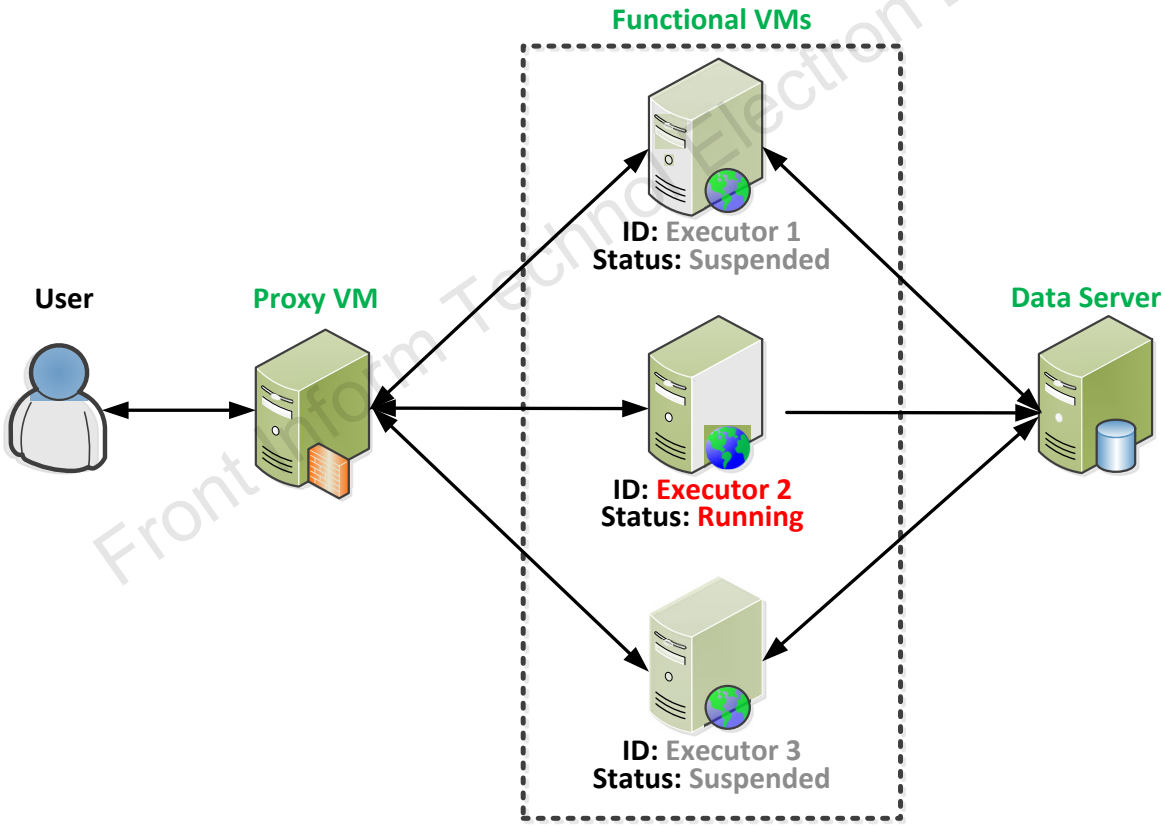
Method (3/9)

New cloud system—overview of Driftor



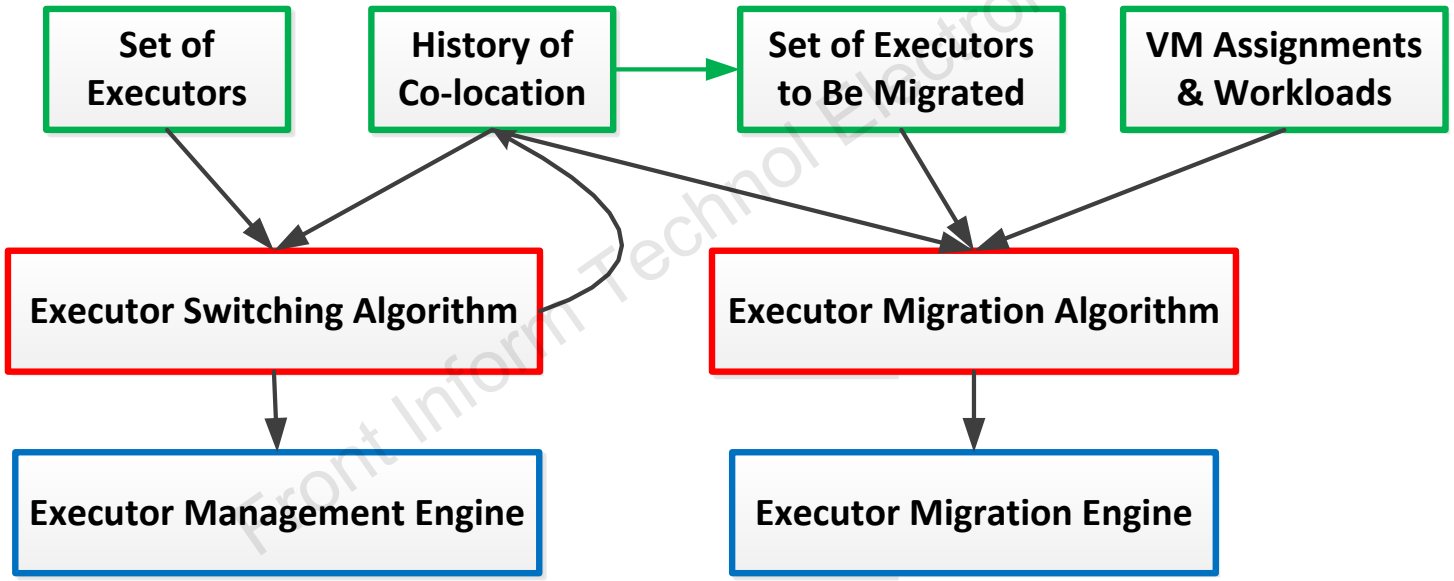
Method (4/9)

New cloud system—multi-executor structure of a virtual machine



Method (5/9)

New cloud system—defense decision engine



Method (6/9)

Defense scheme—symbols and notations

Table 4 Symbols and denotations

Symbol	Meaning	Symbol	Meaning
δ	Security threshold of information leakage attacks	$\mathrm{Loc}(t+\Delta t)=\left\|l_{i,j}^{\mathrm{tar}}\right\|$	Placement of VMs at current epoch
Δt	Defense interval	$\mathrm{NoCo}(t)=\|nc_{i,j}\|$	Client pairs that cannot co-exist at the current epoch
T_{ATT}	Time to fulfill the target side-channel attack	$\mathrm{Co}(t)=\|c_{i,j}\|$	Number of co-resident executors between each pair of clients
$t_{\mathrm{DE}}^{\mathrm{SW}}$ and $T_{\mathrm{OP}}^{\mathrm{SW}}$	Time of a single switch operation and a switch decision, respectively	$\mathrm{NoMig}(t)=\|nm_i\|$	Activated executor for each VM which cannot be migrated in the case of service interruption
$t_{\mathrm{DE}}^{\mathrm{MIG}}$ and $T_{\mathrm{OP}}^{\mathrm{MIG}}$	Time of a migration operation and a migration decision, respectively	$\mathrm{Co}_{\mathrm{up}}(t)=\left\|c_{i,j}^{\mathrm{up}}\right\|$	Number of co-resident active VMs between each pair of clients
N_{C} and N_{S}	Numbers of clients and servers, respectively	$\mathrm{Co}_{\mathrm{up}}^{\mathrm{acc}}(0,\ t)=\left\|c_{i,j}^{\mathrm{acc\text{-}up}}\right\|$	History leakage between each pair of clients
N_{EXE}	Number of executor for each VM	$S_{\mathrm{EXE}}^{i,j}=\left\{e_1^{i,j},\ e_2^{i,j},\ \ldots,\ e_{n\text{-}\mathrm{exe}}^{i,j}\right\}$	Executors of a VM
$N_{\mathrm{S}}^{\mathrm{Cap}}$	Capability of servers to host VMs	$S_{\mathrm{EXE}}=\left\{e_k^{i,j}\right\}$	All VMs in cloud
$N_{\mathrm{C}}^{\mathrm{VM}}=\left\|n_i^{\mathrm{VM}}\right\|,\ i\in[1,\ N_{\mathrm{C}}]$	Number of VMs for each client	$S_{\neg\mathrm{Co}}$ and $S_{\neg\mathrm{Co}}^{t}$	All client-pairs and new client pairs that cannot co-exist
$\mathrm{Loc}(t)=\|l_{i,j}\|$	Placement of VMs at the last epoch		

Method (7/9)

Defense scheme—switching algorithm: switching active state between all executors of a virtual machine

Algorithm 1: *SwitchDecision* that selects active executor for the j-th VM of the i-th client in current defense interval.

Input: $S_{EXE}^{i,j} = \{e_1^{i,j}, e_2^{i,j}, ..., e_{n-exe}^{i,j}\}$ —executors of the j-th VM of the i-th client; $Co_{up}^{acc}(0,t) = \left| c_{i,j}^{acc-up} \right|$ — history co-residency time of different tenants; $S_{\neg Co}$ —client pairs that cannot be co-resident in current interval; $S_{\neg Co}^{t}$ —new client pairs that cannot co-exist; $e_c^{i,j}$ — current active executor.

Output: $e_x^{i,j}$ —executor to be active in current interval

Procedure:

1. $s_c^{i,j} \leftarrow \text{CoresidentVMs}(e_c^{i,j})$;
2. $flag \leftarrow 1$;
3. for each VM x in $s_c^{i,j}$ do:
4. if x is active then:
5. $p \leftarrow \text{MasterTenant}(x)$;
6. if p==i then: continue; end
7. $c_{i,p}^{acc-up} \leftarrow c_{i,p}^{acc-up} + 1$;
8. if $c_{i,p}^{acc-up} \geq k\Delta t$ then:
9. $flag \leftarrow 0$; $S_{\neg Co} \leftarrow S_{\neg Co} + < i, p >$; $S_{\neg Co}^{t} \leftarrow S_{\neg Co}^{t} + < i, p >$
10. end
11. end
12. end
13. $S_{cand-EXE}^{i,j} \leftarrow S_{EXE}^{i,j}$;
14. if flag==0 then: $S_{cand-EXE}^{i,j} \leftarrow S_{cand-EXE}^{i,j} - e_c^{i,j}$ end
15. for each VM x in $S_{cand-EXE}^{i,j}$ do:
16. $y \leftarrow \text{HostServer}(x)$;
17. if isServerExclusive(i, y) then:
18. $S_{cand-EXE}^{i,j} \leftarrow GetVMonServer(i, j, y)$;return;
19. end
20. end
21. $e_x^{i,j} \leftarrow S_{cand-EXE}^{i,j}[\text{rand}(\text{sizeof}(S_{cand-EXE}^{i,j}))]$;

Method (8/9)

Defense scheme—migration algorithm

Intuitive solution: modeling migration-based defense as a satisfiability problem, solved by reducing it to satisfiability for Boolean circuits (CIRCUIT-SAT)

Constraints to be satisfied

$$\forall i \in [1, N_{EXE} \times \sum_{i=1}^{N_C} n_i^{VM}], j \in [1, N_S], l_{i,j}^{tar} \in \{0,1\}$$

$$\forall i \in [1, N_{EXE} \times \sum_{i=1}^{N_C} n_i^{VM}], \sum_{j=1}^{N_S} l_{i,j}^{tar} = 1$$

$$\forall j \in [1, N_S], \sum_{i=1}^{N_{EXE} \times \sum_{i=1}^{N_C} n_i^{VM}} l_{i,j}^{tar} \le N_{CAP}$$

$$\forall i_1, i_2 \in [0, N_C - 1], i_1 \ne i_2, k_1, k_2 \in [1, n_{i_1}^{VM} \times N_{EXE}], k_3 \in [1, n_{i_2}^{VM} \times N_{EXE}],$$

$$k_1 \ne k_2, j \in [1, N_S], l^{tar}_{N_{EXE} \times \sum_{i=1}^{i_1} n_i^{VM} + k_1, j} \times l^{tar}_{N_{EXE} \times \sum_{i=1}^{i_1} n_i^{VM} + k_2, j} \times l^{tar}_{N_{EXE} \times \sum_{i=1}^{i_2} n_i^{VM} + k_3, j} = 0$$

$$\forall i_1, i_2 \in [0, N_C - 1], i_1 \ne i_2, c_{i_1,i_2} = \sum_{j=1}^{N_S} \sum_{k_1=1}^{n_{i_1}^{VM} \times N_{EXE}} \sum_{k_2=1}^{n_{i_2}^{VM} \times N_{EXE}} l^{tar}_{i_1 \times N_{EXE} + k_1, j} \times l^{tar}_{i_2 \times N_{EXE} + k_2, j}$$

$$\forall i_1, i_2 \in [0, N_C - 1], i_1 \ne i_2, c_{i_1,i_2} \le nc_{i_1,i_2}$$

Bad scalability

Method (9/9)

Defense scheme—migration algorithm

GreedyLikeHeuristic: search a viable solution by gradually expanding an initial has-to-migrate set of VMs

Algorithm 2: *GreedyLikeHeuristic* that tries to find minimal migration solution in the huge search space

Input: $Loc(t) == \left|l_{i,j}\right|$ —placement strategy in last defense interval; $S_{EXE} = \{e_k^{i,j}\}$ —all executors in cloud; $S_{\neg Co}$ —VM pairs that cannot be co-resident in current interval; $S_{\neg Co}^{t}$ —new client pairs that cannot co-exist; $e_c^{i,j}$ — current active executor; $Loc(t) == \left|l_{i,j}\right|$—placement in last defense interval.

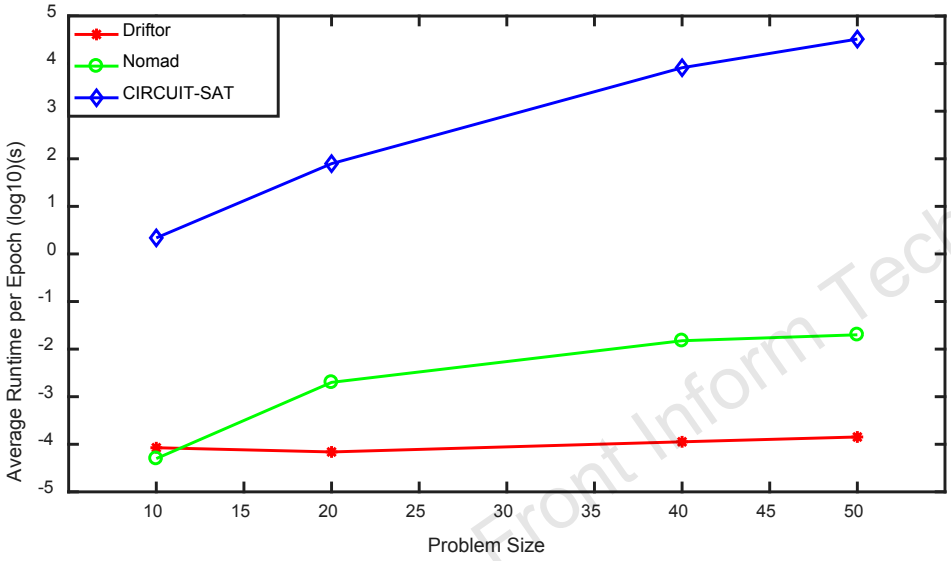
Output: $Loc(t+\Delta t) == \left|l_{i,j}^{tar}\right|$ —new placement for current defense interval; Error—no satisfactory solution is found.

Procedure:

1. $S_{\neg Co}^{\det} = \{< e_{k1}^{i1,j1}, e_{k2}^{i2,j2}, s >\} \leftarrow \text{detailUncolPairs}(S_{\neg Co}^{t}, Loc(t))$;
2. $S_{\neg Co}^{\det-S} = \{< s, \{< e_{k1}^{i1,j1}, e_{k2}^{i2,j2} >\} >\} \leftarrow \text{groupByServer}(S_{\neg Co}^{\det})$;
3. $S_{Mig}^{\min-S} = \{< s, \{\{e_k^{i,j}\}\} >\} \leftarrow \text{calMinMigSets}(S_{\neg Co}^{\det-S})$;
4. $N_{Mig}^{Init} = \text{numInitialMigSets}(S_{Mig}^{\min-S})$;
5. **for** $p \leftarrow 1$ to N_{Mig}^{Init} **do**:
6. $S_{Mig}^{Init} = \{e_k^{i,j}\} \leftarrow \text{pickInitialMigSet}(S_{Mig}^{\min-S}, p)$;
7. $(numMigClients, S_{Mig}^{Init-C} = \{< i, \{e_k^{i,j}\} >\}) \leftarrow \text{groupByClient}(S_{Mig}^{Init})$;
8. $S_{Mig}^{Init-C-Sort} = \{< i, \{e_k^{i,j}\} >\} \leftarrow \text{sortByClient}(S_{Mig}^{Init-C}, S_{\neg Co})$;
9. **for** $n_C \leftarrow 1$ to *numMigClients* **do**:
10. $clientId \leftarrow (S_{Mig}^{Init-C-Sort} \rightarrow at(n_C).i)$;
11. place as many executors of *clientId* as possible on its exclusive servers;
12. **if** all executors of *clientId* has been assigned with new servers **then do**: **continue**; **end**
13. place as many left executors of *clientId* as possible on empty servers;
14. **if** all executors of *clientId* has been assigned with new servers **then do**: **continue**; **end**
15. **for each** server *se* **in** left servers **do**:
16. **if** *se* is fully occupied **then do**: **continue**; **end**
17. **if** *se* is exclusive for client other than *clientId* **then do**: **continue**; **end**
18. **if** *se* already hosts an executor of *clientId* **then do**: **continue**; **end**
19. $S_{\neg Col}^{clientId} = \{e_k^{i,j}\} \leftarrow \text{uncolExes}(clientId, S_{\neg Col})$;
20. **if** $\text{sizeof}(S_{\neg Col}^{clientId}) > 1$ **then do**: **continue**;
21. **else if** $\text{sizeof}(S_{\neg Col}^{clientId}) == 1$ **then do**:
22. **if** $S_{\neg Col}^{clientId} \rightarrow at(0).i$ can co-exist with all other executors on the same server as any unassigned executor $e_k^{i,j}$ of clientId **then do**:
23. switch $S_{\neg Col}^{clientId} \rightarrow at(0).i$ with $e_k^{i,j}$;
24. **end**
25. **else** (if $\text{sizeof}(S_{\neg Col}^{clientId}) == 0$) **then do**:
26. switch one executor of *clientId* with randomly selected executor on *se*;
27. **end if**
28. **if** all executors of *clientId* has been assigned with new servers **then do**: **break**; **end**
29. **end for**
30. **end for**
31. **if** $n_C > numMigClients$ **then do**:
32. Output $Loc(t+\Delta t) == \left|l_{i,j}^{tar}\right|$;
33. **return**;
34. **end**
35. **end for**
36. **print** Error;

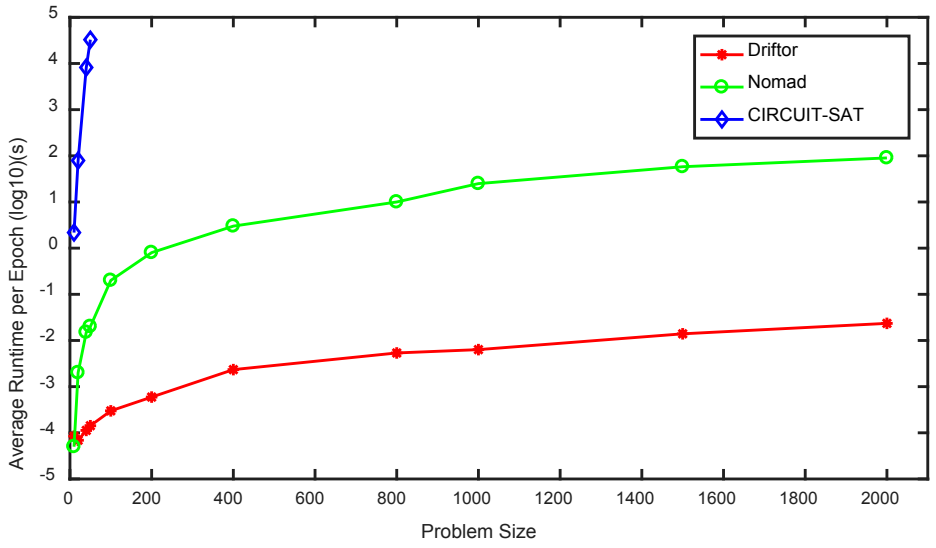
Major results (1/6)

Scalability test—compare Driftor's migration algorithm with Nomad and CIRCUIT-SAT for their scalability to solve the migration problem



Problem size ranging from 10 to 50

Problem size ranging from 10 to 2000



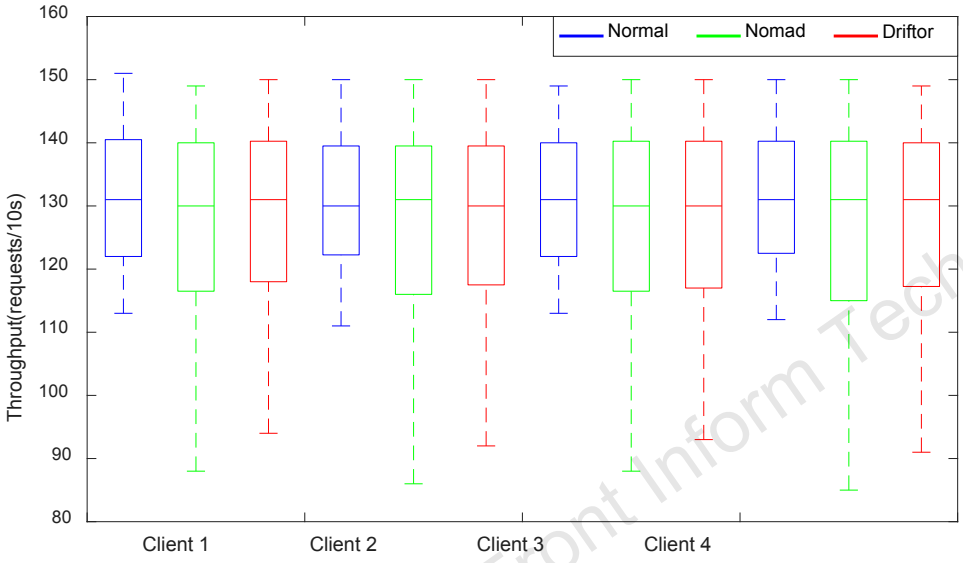
Major results (2/6)

Real defense overhead—switching and migration time of different instances

Type of instance	T_{OP}^{SW} (s)	T_{OP}^{MIG} (s)
Ubuntu-Cloud (512 MB RAM, 1.5 GB)	1.04	2.12
Cirros (512 MB RAM, 132 MB)	0.77	0.41
Ubuntu (2048 MB RAM, 7 GB)	1.26	8.09

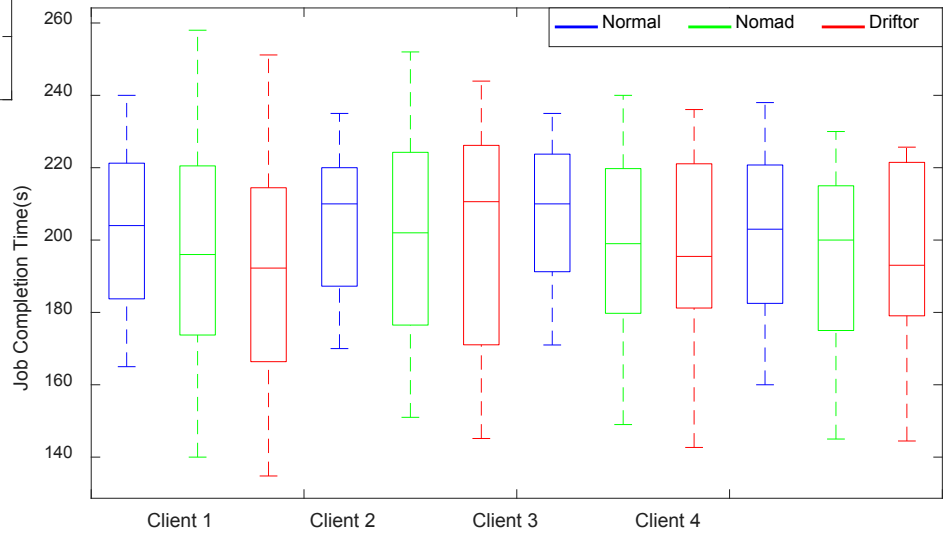
Major results (3/6)

Real defense overhead—impact on practical cloud-based applications



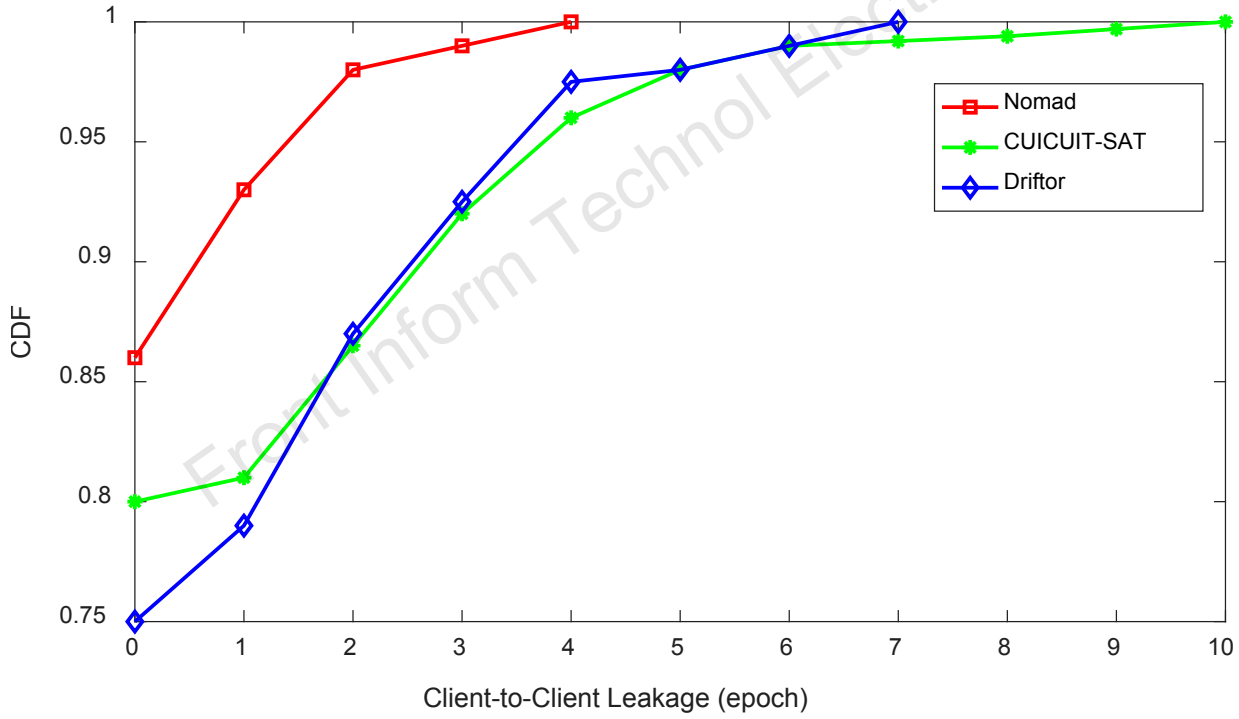
Web-server evaluation

MapReduce evaluation



Major results (4/6)

Security analysis—compare Driftor's information leakage with Nomad and CIRCUIT-SAT



Major results (5/6)

Security analysis—evaluate co-residency time of different tenant pairs under Driftor

	1		10		14		17	
3	1	0	4	0	5	0	0	0
8	0	0	0	0	2	0	3	0
16	3	0	0	0	0	0	0	0
18	2	0	4	0	0	0	1	0

Major results (6/6)

Security analysis—compare Driftor with pure switch for their defense ability

	1		10		14		17	
3	0	0	4	0	7	0	0	0
8	0	0	0	0	0	0	4	0
16	3	0	0	0	0	0	0	0
18	3	0	6	0	0	0	0	0

Conclusions

Driftor can not only defend against practical fast side-channel attacks, but also introduce reasonable impact on real-world cloud applications.