Diverse Infrastructure and Architecture for Datacenter and Cloud Resilience

James P.G. Sterbenz and Prasad Kulkarni
Information and Telecommunication Technology Center
Department of Electrical Engineering and Computer Science
The University of Kansas, Lawrence, Kansas, 66045, USA
{jpgs, kulkarni}@ittc.ku.edu
www.ittc.ku.edu/resilinets

***Abstract*—Internet and web services have seen widespread adoption in recent years and are now tightly integrated into society's daily activities. An important emerging part of the Internet is *clouds* that provide low-cost configurable computing resources, allowing businesses to reduce their hardware, software, and personnel costs. Increasingly, enterprises now use such cloud resources to host web applications. While clouds provide an excellent business model, most existing public and private cloud infrastructures are based on *monocultures* that allow attackers to focus their efforts on a single hardware/software platform and facilitates the rapid spreading of successful attacks.**

In this invited paper, we describe a methodology and mechanisms that make *clouds and hosted applications considerably more resilient to attacks and correlated failures by introducing diversity at every level of the cloud*: physical interconnect, network components, processor platforms, storage management, virtual machine monitors, operating systems, and application processes. Our goal is to defend against attacks by continuing to operate correctly even when part of the infrastructure fails and to substantially raise the difficulty of executing a successful attack by requiring the attacker to simultaneously target different hardware and software choices. Furthermore, by geographically spreading applications among different datacenters using diverse network connections (in service provider and access medium – wired vs. wireless), the cloud will be resilient against physical infrastructure attacks and large-scale disasters.

***Index Terms*—datacenter cloud network, resilient survivable fault-tolerant, diverse agile, moving-target defence**

I. Introduction and Motivation

With increasing dependence on Internet services to support society's daily activities, the consequences of disruption from an attack or disaster are increasingly severe [1]. An important emerging part of the Internet are *clouds* from which users subscribe to shared configurable computing resources. While convenient, private and public clouds are currently based on commodity monocultures through which attacks can rapidly spread. Furthermore, clouds and their access may not be sufficiently distributed to survive infrastructure attacks.

The *DefCloud* architecture is intended to make clouds considerably more resilient to attacks and correlated failures by introducing diversity at every level of the cloud: physical interconnect, network components, processor platforms, storage management, virtual machine monitors, operating systems, and application processes. The goal is to *defend against attacks* by continuing to operate even when part of the infrastructure fails and to substantially *increase the difficulty of executing a successful attack* by requiring the attacker to simultaneously target different hardware and software choices. Furthermore, by *geographically spreading applications among different datacenters using diverse network connections* (in service provider and access medium – wired vs. wireless), the cloud will be resilient against physical infrastructure attacks and large-scale disasters, such as power failures, hurricanes, earthquakes, CMEs (coronal mass ejections) and EMP (electromagnetic pulse) weapons. Finally, cloud applications are made more resilient by *spreading in both space and time* across the cloud and datacenter diversity.

The rest of the paper is arranged as follows. Section II describes the DefCloud diverse datacenter and cloud infrastructure that is resilient and survivable to attacks and large-scale disasters resulting in geographically-correlated failures. Section III describes DefCloud program diversity in space and time that increases defense against attacks on the software running on the cloud. Section IV describes our approach to measure the resilience delivered by the DefCloud approach. Finally, Section V provides a brief summary of this paper and suggests directions for further research.

II. Infrastructure-Level Diversity

The first major aspect of diversity is in the datacenter and cloud infrastructure, consisting of both physical hardware as well as systems software. This section first describes DefCloud datacenter diversity to resist attacks against particular hardware and software platforms. Then diversity in the cloud deployment is described to provide resilience against the geographically-correlated failures that result from large-scale disasters such as hurricanes, coronal mass ejections, EMP (electromagnetic-pulse) weapons, and large-scale power blackouts.

A. Datacenter Diversity

The first major aspect of diversity is in the datacenter infrasructure, consisting of both physical hardware as well as systems software, as shown in Figure 1. The upper layers of the diagram are core (C), aggregation (A), and edge (E) switches, typically using Gigabit Ethernet in the lower layers and 10Gig Ethernet in the upper layers to the core switches

978-1-4673-2178-5/12/$31.00

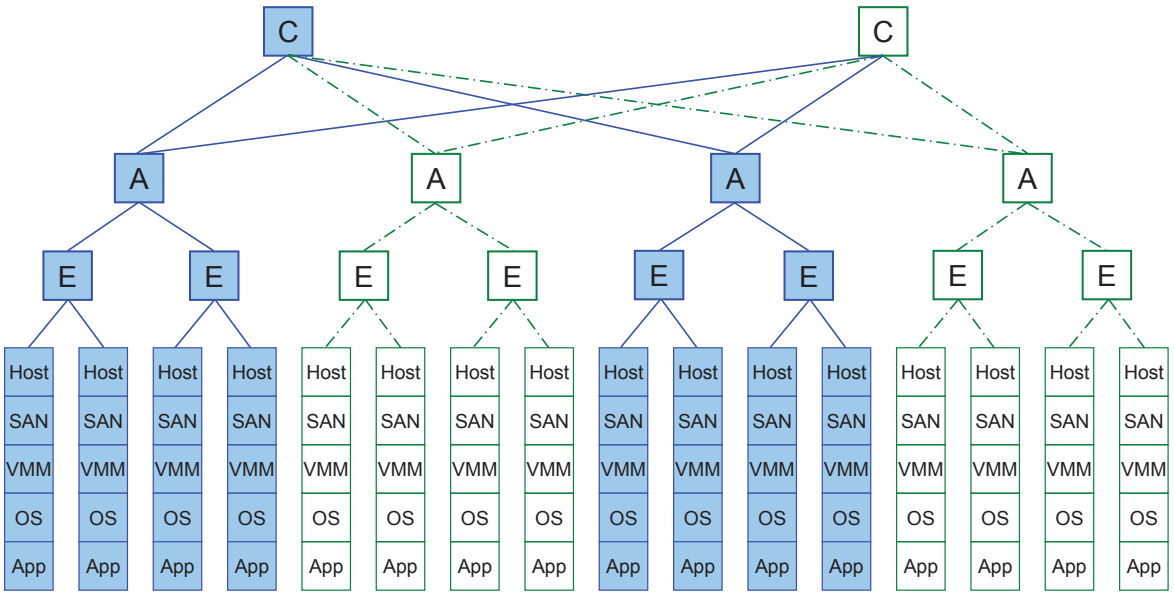

Fig. 1. Datacenter architecture

and external Internet access links. The network interconnection shown is typical cross-connected trees to provide redundancy and load balancing capabilities, in a fat-tree or Clos connection [2], [3]. Commodity switches from a single vendor are typically used in existing clouds to achieve economies of scale and reduce costs, including datacenter management costs.

At the lowest layer are the processors, consisting of a computer platform (Host), with storage management (SAN), running a virtual machine monitor or hypervisor (VMM), running a guest operating system (OS), on which application processes run (App). As with the network infrastructure, a single set of choices is made to reduce costs for processor vendor, storage management solution, virtual machine hypervisor, and operating system. Such an implementation results in a monoculture through which malware can quickly spread. A promising solution is to add diversity to eliminate the monoculture over which application processes are spread, with the understanding that this comes at the cost of additional management overhead and redundant infrastructure. The premise of DefCloud is that *redundancy and diversity are necessary for resilience*, with flexible alternatives to balance cost- and management-effective deployment alternatives, while also letting applications specify the needed level of resilience so that we can dynamically adapt the cloud datacenters in a service-aware manner.

The most straightforward way to introduce diversity into the datacenter is to divide it into k subtreees; Figure 1 shows a case for $k = 2$ in which one subtree is denoted by shaded boxes and solid line interconnects and the other uses white boxes with dashed interconnects. The idea is to choose among a set of choices at each level and assign a single choice to each subtree such that any network-induced malware can at most affect one subtree. For example, one subtree might use Cisco switches for the network infrastructure and IBM processors with QLogic storage management, running Xen hypervisors with Linux operating systems. The other subtree might use Juniper switches for the network infrastructure and HP processors with LSI Logic storage management, running VMware hypervisors with Windows operating systems. Thus any attack against a platform's BIOS, firmware, or software exploits will be confined in damage. This will be true even for zero-day attacks for which conventional security software fails. This approach has two significant advantages:

1) the probability of malware attacks bringing down the entire datacenter is significantly lowered
2) the bar has been significantly raised for attackers by requiring simultaneous exploits across different architectures (e.g. Linux and Windows)

B. Cloud Diversity

While spreading across k subtrees protects against software- and firmware-based attacks, it does not protect against attacks or disasters that destroy or disable the physical infrastructure. While datacenters are typically fortified, guarded, and have significant power backup, they are not immune from terrorist attacks, EMP weapons, nor the results of natural disasters such as hurricanes, earthquakes, tsunami, or infrastructure interdependencies such as large-scale long-lived power blackouts. In this case the solution is to distribute a datacenter [4] over n geographically distributed locations, as shown in Figure 2.

In this case n datacenters are connected by multi-homing the core switches to different ISPs, with maximum protection obtained if there is diversity in the access link technology, for example fiber to one ISP and a high bandwidth wireless point-to-point link to the other. It is still possible that the links within service providers share the fate of carrier hotel co-location or physical fiber conduit (the latter happened in the Baltimore tunnel fire [5], [6]); multilayer geographic diversity

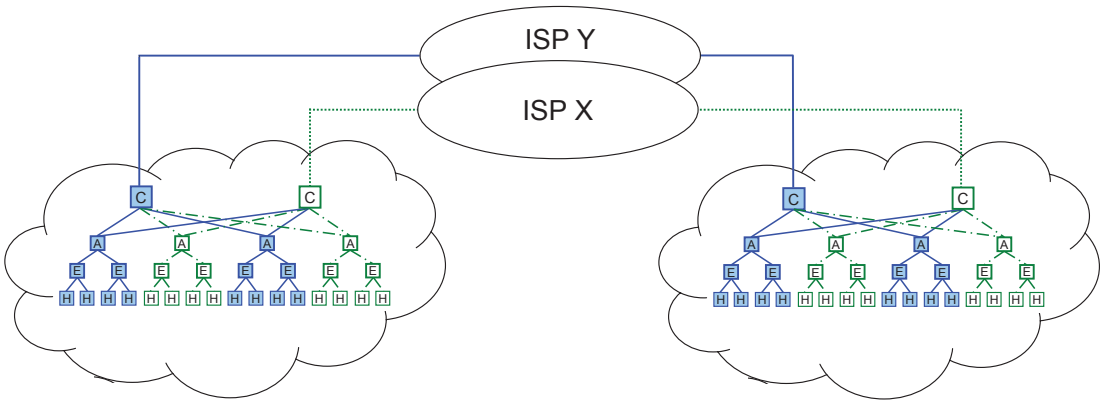

Fig. 2. Cloude diversity and datacenter interconnectivity

for resilience [7], [8] is necessary to ensure that multiple paths do not share geographic fate.

Figure 3a shows an example of the area that might be affected by an EMP (electromagnetic pule) weapon over New York; in this case we would want a distributed cloud to also have a datacenter in at least the midwest or southeast US. Figure 3b shows an example of an area covered by increasingly severe power blackouts in the midwest US. These examples are intended to show our capability to model a wide range of attacks and geographic threats; details and example simulation runs are presented in [9].

III. Process-Level Program Diversity

This section describes process-level diversity in space and time. The user of a cloud computing environment often has the most control over the configuration of the application program. Nevertheless, existing software development and execution models statically construct and ship a single program binary for execution. This static model is convenient for software developers, who now only need to build, optimise, debug, and maintain one version of each program. Unfortunately, this model also permits attackers sufficient time to carefully study one static binary, and discover and exploit vulnerabilities in the software. At the same time, a single attack vector for a particular software program is able to uniformly affect all program copies running on computers all over the world. Thus, this software monoculture is, at least partially, responsible for the prevalence of software attacks and the huge resulting financial and privacy losses [10].

Process-level program diversity is an approach to significantly mitigate the issue of software monoculture. We are designing and implementing new techniques to evaluate the impact of process diversity along two complementary domains – *space* and *time*. Process diversity in the *space* domain entails initiating multiple distinct program configurations on different cloud nodes. Discrepancies in the program state calculated by the different program versions indicate the possibility of attacks. In contrast, process diversity in the *time* domain requires the runtime system to dynamically alter the process configuration periodically or at certain enabling events. Frameworks that provide such time-based diversity are also called *moving-target defense* (MTD) systems. Our mechanisms are designed to achieve better flexibility and diversity at lower costs than existing schemes.

A. Process Diversity in Space

One approach to thwart attacks is to spawn multiple simultaneous executions of *distinct* variants of the same software task and compare their results for consistence. The challenge in this approach is to create program variants that are not susceptible to the same attack without affecting code development practices, software usability, execution efficiency, or cloud system administration. Existing program diversification techniques fail in this regard. Our approach achieves program diversity by partitioning the program horizontally into multiple independent code *slices* [11]. A program slice contains only those program statements that are relevant to a particular computation statement, called a *slicing criterion*.

We generate program slices such that each slice only contains a subset of the original program instructions and data structures necessary to compute different *partial* program states. Consequently, attacks customised for any one program layout fail (with a high probability) for the other program slices. The output of each program slice is compared with the corresponding program output to detect attacks. Slices are automatically generated by the compiler. It is also possible to customise the number and size of program slices based on available computing resources.

Our compiler analyses the input program and produces a *program dependence graph* (PDG) [12], to manifest both the control and data dependences for each operation in the program. For example, Figure 4 illustrates a sample program along with its program dependence graph. Solid edges in the figure mark data dependences, while the dotted edges show the control dependence between pairs of program instructions. The compiler uses the PDG to detect independent data flows in different regions of the program.

Next, we employ different mechanisms to identify sets of *slice criteria* to generate slices that achieve effective code diversity. These mechanisms include conservative techniques

(a) EMP weapon (b) Power blackout

Fig. 3. Infrastructure challenge examples

for automatic partitioning of a PDG into parallel tasks [13] as well as aggressive slicing and consolidation of slices for all *important* program variables or live ranges. For example, a slicing criteria consisting of important variables for the program in Figure 4 will contain instructions S8, S9 and S10 corresponding to variables `result`, `freqRenterPts`, and `tAmount` respectively.

The slicing criteria is used by the slicer to partition the program into smaller *executable* slices. Figure 5 shows the slices that are created for the program in Figure 4 given the above slicing criteria (statements S8, S9, and S10).

Finally, the primary program thread is instrumented with additional instructions to compare its program state at appropriate intermediate points with the state computed by each distinct program slice. The compiler generates the code to communicate and compare program states, and take appropriate action based on the comparison outcome.

Providing an *N*-Version Execution Environment: The criticality of the user task in combination with the cost and availability of cloud resources guides the level and amount of redundancy provided by the system. Thus, for a 2-level redundancy scheme, the process diversity algorithm generates program slices such that, in addition to the original unmodified program, all process state is generated by *at least* one slice, and *at most* one slice, whenever feasible. At the same time, this approach is powerful enough to generate multiple distinct program slices that actively attempt to (re)compute the same program state for cases when a greater level of redundancy may be necessary for more critical tasks.

In contrast to previous *N*-version programming techniques [14], [15], the DefCloud approach requires no hardware changes. Program slices only compute a subset of program state, enabling them to generally run faster than the original program. At the same time, since each generated program slice has a largely different code and data layout, we expect our technique to provide the same or better resilience than existing software diversity schemes.

B. Process Diversity in Time – Moving Target Defense

Compile and load-time diversity forces attackers to tailor attacks to each software variant, but do not prevent attacks on a single process. Run-time diversity presents attackers with a *moving target*, making it extremely difficult to compromise the system based on knowledge of the program's a priori vulnerabilities. DefCloud uses a virtualisation based solution to periodically vary the executed code and data layout. Program diversity is achieved by randomizing the code generation and optimisation aspects of our just-in-time (JIT) compiler.

Native binary executables generally do not retain enough high-level syntax and semantic information to allow the most effective randomization schemes to be applied at runtime, resulting in weak protection against attacks [16], [17]. Therefore, our virtualisation-based runtime environment employs a powerful binary decompiler to recover semantic program information and enable high-level randomisation techniques. The decompiler uses a combination of static and incremental dynamic translation to convert the original binary executable into our compiler's higher-level intermediate representation at runtime [18].

DefCloud utilises a *staged* model for applying code diversification transformations to minimise startup overhead. Fast, but lower-entropy, randomisation techniques are applied in the first stage. The later stages employ deep static analysis to discover high-level program information (including function definitions, arguments, and variable types) and enable fine-grained compiler transformations to diversify each code block. We are developing a state search space evaluator to assist in code re-randomisation decisions.

Figure 6 illustrates the architecture of the DefCloud MTD framework. The original application binary is handed to the virtual machine (VM), which mediates program execution by periodically diversifying each program block before it is run to provide a continuously varying surface to malicious attackers. The DefCloud framework includes a decompiler to read the input binary and translate it into the *intermediate code* (IR) for our compiler. Decompilation is conducted in multiple stages, so that the first stage only produces a naive (fast) translation. Later stages discover and recover finer and higher-level information about the program.

Several randomising transformations are activated at runtime during the intermediate (IR) to binary code translation. The activated transformations depend on the amount of high-level program information made available to the compiler. DefCloud also investigates schemes to characterise the pro-

```
S0:  public String statement() {
S1:     double tAmount = 0;
S2:     int freqRenterPts = 0;
S3:     String result = "Record for " +
           getName() + "\n";
S4:     Iterator rentals = _rent.iterator();
S5:     while (rentals.hasNext()) {
S6:        Rental each = rentals.next();
S7:        result += "\t" + each.getTitle();
S8:        result += "\t" + each.getCharge();
S9:        freqRenterPts += each.getFRP();
S10:       tAmount += each.getCharge();
        }
S11:    String final = result + tAmount +
                       freqRenterPts;
S12:    return final;
     }
```

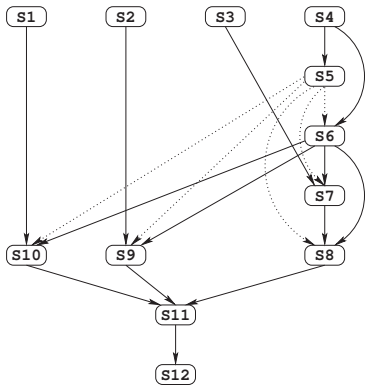

Fig. 4. Sample program and corresponding program dependence graph

```
S0: public String statement_slice1() {

S3:   String result = "Record for " +
         getName() + "\n";
S4:   Iterator rentals = _rent.iterator();
S5:   while (rentals.hasNext()) {
S6:     Rental each = rentals.next();
S7:     result += "\t" + each.getTitle();
S8:     result += "\t" + each.getCharge();

      }

    }
```

```
S0: public String statement_slice2() {

S2:   int freqRenterPts = 0;

S4:   Iterator rentals = _rent.iterator();
S5:   while (rentals.hasNext()) {
S6:     Rental each = rentals.next();

S9:     freqRenterPts += each.getFRP();

      }

    }
```

```
S0: public String statement_slice3() {
S1:   double tAmount = 0;

S4:   Iterator rentals = _rent.iterator();
S5:   while (rentals.hasNext()) {
S6:     Rental each = rentals.next();

S10:    tAmount += each.getCharge();
      }

    }
```

Fig. 5. Program slices corresponding to the statements S8, S9, and S10 respectively of the example program in Figure 4

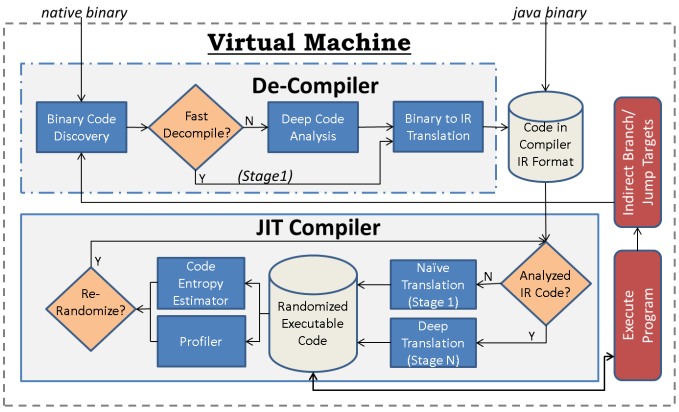

Fig. 6. Virtual Machine Framework to Provide MTD for Binary Applications

gram state search space to understand the effect of each randomisation technique on program entropy. This is used to guide the timing of re-randomisation for each code block, as well as the selection and intensity of future program transformations for software diversity. To achieve high performance, DefCloud initiates several compiler threads to translate code speculatively and in parallel before it is demanded by the application thread(s).

Program diversity in existing systems has been accomplished by several techniques, including popular randomisation schemes such as *address space layout randomisation* (ASLR) [19] and *instruction set randomisation* (ISR) [20]. However, existing techniques are plagued by their susceptibility to brute-force attacks, low randomisation entropy, and poor performance.

The DefCloud framework for an MTD system is unique in its approach, capabilities, implementation, and resulting efficiency. First, reconstructing higher-level program information allows our framework to apply a range of low-level diversity transformations that are off-limits to many existing run-time randomisation tools. Second, DefCloud does not require any changes to the static binary executable, but relys solely on runtime transformations to achieve code diversity. Third, we are experimenting with techniques to automatically determine the timing of future re-randomisations for each code block. Finally, DefCloud is be designed to employ a *staged* compilation model and exploit multi-core processors to ensure high system performance.

IV. Resilience Evaluation

The previous sections describe how to create and deploy diversity in hardware and systems-software infrastructure and cloud applications. Such diversity in the execution environment leads to the problem of ideally assigning processes to this infrastructure, routing information between them, and cross-layering service awareness between the infrastructure and cloud applications. This section describes the planned application of our ResiliNets evaluation methodology to DefCloud.

The DefCloud (n, k) infrastructure diversity provides n datacenters containing k subtrees. The maximum resilience comes at the greatest cost: completely replicating application

processes across subtrees spread across datacenters. While 2-redundancy is almost certainly sufficient for an order of magnitude improvement in resiliency, 3-redundancy is required for triple-modular redundant (TMR) voting schemes [21]. In this case, assuming that malware can spread across the Internet between the interconnected datacenters, we need $k = 3$ subtree architectural alternatives with each in a different datacenter for $n = 3$. This permits high resiliency of service- and time-critical applications that cannot afford to stop running. A spectrum of options exists including monitoring application progress and reallocating processes to alternative architecture trees at the cost of the migration delay. The process-level space- and time-spreading techniques described in Section III are used to duplicate process state computation by diverse processes on multiple nodes, and to continually randomise the process code and data layout on each node. Furthermore, we need to understand the best way of routing within and across subtrees and data centers, and the APIs and cross-layer mechanisms to allow applications to express their resilience needs to affect process allocation and datacenter logical topology, and for the infrastructure to express its state to the process spreading algorithms.

A. Analysis Methodology

A key part of the DefCloud approach is to try alternative diversity, redundancy, and spreading strategies, with respect to both the network infrastructure and process allocation, to understand their resilience. This must be done given a variety of threat models, and we put particular emphasis on: (1) zero-day malware attacks that will cause all instances of a particular type of hardware or systems software components to fail, and (2) physical attacks or large-scale disasters that will cause an entire datacenter to fail or its interconnection with the Internet and other datacenters to fail.

One of our goals is to understand the optimal strategy to use given these challenges and the service-oriented resilience requirements, with the understanding that there are diminishing returns to increasing diversity and redundancy beyond a certain level. Our earlier research studies a wide range of network topologies and shows that the greatest benefit is obtained with a small number of diverse options (two or three) [8].

It is important to have a measure for resilience, but a single resilience metric is not obvious given the number of parameters and variables involved. We have developed a novel resilience metric computed as the area under the trajectory through a two dimensional state space [22]–[25]. The horizontal axis is a functional combination of several parameters that indicates the operational state of the network; in the case of DefCloud, it reflects to what degree datacenters or parts thereof within the distributed cloud have failed. The vertical axis reflects the service provided, in our case the ability of the application processes to continue to execute as required. As the infrastructure is challenged, the trajectory moves from *normal operation* through *partially degraded* to *severely degraded*. We dimension DefCloud diversity and redundancy such that the infrastructure is at most partially degraded. As the network degrades, we need to understand how resilient the applications are in the face of that degradation. Thus a more resilient application will show a shallower slope in the trajectory, and we dimension DefCloud application spreading to be no worse than normal or partially degraded operation, depending on the user requirements for application resilience.

We need a way to simulate and emulate (for various challenges to the cloud, in particular zero-day and physical infrastructure attacks. We have developed a novel simulation methodology that allows us to apply various *challenges* to a variety of network scenarios [9], [26]. Our methodology allows the specification of component failures, either random, or based on some criteria such as type (which we would apply to datacenter components subject to a zero-day attack) or graph-theoretic criticality (applied to an attacker who understands the structure of the network to destroy key components). We can also specify area-based challenges using a circle or arbitrary polygon overlaid on the network geography; these can evolve and move over time.

Both space-domain and time-domain process-level diversity techniques will be implemented and integrated with this simulation methodology to evaluate their impact individually and in combination with the other diversity mechanisms at the network, hardware, hypervisor, and OS levels. We will build process-level models to analyse and understand the costs, benefits and diminishing returns to process resilience as we intensify the application of our process-level diversity techniques. The traffic generated by each deployed process will be modeled, and handed to the remaining analysis framework to study process performance under various attack scenarios to determine the optimal diversification strategy to adopt given resilience requirements.

Finally, we will deploy prototype DefCloud code in a testbed environment. For this we will use the existing GpENI global programmable testbed [27], [28] enhanced with ProtoGENI/Emulab clusters. GpENI provides a programmable testbed on which various topologies can be instantiated and components taken down to emulate failures. GpENI allows us to emulate geographic distribution of datacenters. ProtoGENI clusters are GENI-version of Emulab that are local machine clusters with multiple network interfaces that can be configured in a variety of ways; ideal for emulating datacenters.

Our challenge framework will be extended to process-level failures [9], [26], and will be used to emulate various attacks to the cloud and understand requirements for meeting the goals for process-level diversity. Software attacks on the space-domain techniques will also be emulated by randomly halting or altering the output of some task processes. Benchmark programs will be statically updated to introduce vulnerabilities, and memory attacks will be introduced during run-time tasks to explore and evaluate the effectiveness of our techniques. Such vulnerable processes build and executed on run-time systems configured with DefCloud diversity techniques will be deployed in a data center using ProroGENI/Emulab cluster in a GpENI based testbed. Observed effects of software attacks on the cost and achieved resilience properties of processes under

different configurations of alternative diversity techniques will be measured. The time-domain diversity schemes will also be measured based on the state space of randomisation that we will be able to generate with our set of diversification techniques [17].

V. Conclusions and Future Work

This paper has described the DefCloud approach to make clouds and hosted applications considerably more resilient to attacks and correlated failures by introducing diversity at every level of the cloud: physical interconnect, network components, processor platforms, storage management, virtual machine monitors, operating systems, and application processes. In particular, DefCloud provides geographically diverse infrastructure the resists attacks against a monoculture and area-based challenges resulting from large-scale disasters. Furthermore, by providing process-level diversity in space and time, an attacker would have to simultaneously target different hardware and software choices, and constantly adapt the attack.

We intend to simulate and emulate the DefCloud architecture to evaluate its effectiveness and understand the cost-benefit tradeoffs to achieve desired levels of resilience.

References

[1] J. P. G. Sterbenz, D. Hutchison, E. K. Çetinkaya, A. Jabbar, J. P. Rohrer, M. Schöller, and P. Smith, "Resilience and survivability in communication networks: Strategies, principles, and survey of disciplines," *Computer Networks*, vol. 54, no. 8, pp. 1245–1265, 2010.

[2] M. Al-Fares, A. Loukissas, and A. Vahdat, "A scalable, Commodity Data Center Network Architecture," in *Proceedings of the ACM SIGCOMM Conference on Data Communication*, (Seattle, WA, USA), pp. 63–74, 2008.

[3] A. Greenberg, J. R. Hamilton, N. Jain, S. Kandula, C. Kim, P. Lahiri, D. A. Maltz, P. Patel, and S. Sengupta, "VL2: A Scalable and Flexible Data Center Network," in *Proceedings of the ACM SIGCOMM Conference on Data Communication*, (Barcelona, Spain), pp. 51–62, 2009.

[4] R. Cocchiara, H. Davis, and D. Kinnaird, "Data center topologies for mission-critical business systems," *IBM Syst. J.*, vol. 47, pp. 695–706, October 2008.

[5] M. R. Carter, M. P. Howard, N. Owens, D. Register, J. Kennedy, K. Pecheux, and A. Newton, "Effects of catastrophic events on transportation system management and operations, Howard Street tunnel fire, Baltimore City, Maryland – July 18, 2001," tech. rep., U.S. Department of Transportation, ITS Joint Program Office, Washington DC, 2002.

[6] H. C. Styron, "CSX tunnel fire: Baltimore, MD," US Fire Administration Technical Report USFA-TR-140, Federal Emergency Management Administration, Emmitsburg, MD, 2001.

[7] J. P. Rohrer, A. Jabbar, and J. P. G. Sterbenz, "Path diversification: A multipath resilience mechanism," in *Proceedings of the IEEE 7th International Workshop on the Design of Reliable Communication Networks (DRCN)*, (Washington, DC), pp. 343–351, October 2009.

[8] J. P. Rohrer and J. P. G. Sterbenz, "Predicting topology survivability using path diversity," in *Proceedings of the IEEE/IFIP International Workshop on Reliable Networks Design and Modeling (RNDM)*, (Budapest), pp. 95–101, October 2011.

[9] E. K. Çetinkaya, D. Broyles, A. Dandekar, S. Srinivasan, and J. P. G. Sterbenz, "Modelling Communication Network Challenges for Future Internet Resilience, Survivability, and Disruption Tolerance: A Simulation-Based Approach," *Springer Telecommunication Systems*, pp. 1–16, 2011. Published online: 21 September 2011.

[10] S. Forrest, A. Somayaji, and D. Ackley, "Building diverse computer systems," in *Proceedings of the 6th Workshop on Hot Topics in Operating Systems (HotOS-VI)*, (Washington, DC, USA), pp. 67–, IEEE Computer Society, 1997.

[11] M. Weiser, "Program slicing," in *ICSE '81: Proceedings of the 5th international conference on Software engineering*, (Piscataway, NJ, USA), pp. 439–449, IEEE Press, 1981.

[12] J. Ferrante, K. J. Ottenstein, and J. D. Warren, "The program dependence graph and its use in optimization," *ACM Trans. Program. Lang. Syst.*, vol. 9, pp. 319–349, July 1987.

[13] V. Sarkar, "Automatic partitioning of a program dependence graph into parallel tasks," *IBM J. Res. Dev.*, vol. 35, pp. 779–804, September 1991.

[14] A. A. Avizienis and M. R. L. (Editor), *Chapter-2: The Methodology of N-version Programming in Software Fault Tolerance*. John Wiley & Sons Inc; 1 edition, 1995.

[15] A. Singh, N. Sinha, and N. Agrawal, "Avatars for pennies: Cheap n-version programming for replication," in *Short paper in the 6th Workshop on Hot Topics in System Dependability (HotDep '10)*, pp. 1–3, October 2010.

[16] H. Shacham, M. Page, B. Pfaff, E.-J. Goh, N. Modadugu, and D. Boneh, "On the effectiveness of address-space randomization," in *Proceedings of the 11th ACM conference on Computer and communications security*, pp. 298–307, 2004.

[17] A. Nguyen-Tuong, A. Wang, J. D. Hiser, J. C. Knight, and J. W. Davidson, "On the effectiveness of the metamorphic shield," in *Proceedings of the Fourth European Conference on Software Architecture: Companion Volume*, pp. 170–174, 2010.

[18] S. Nanda, W. Li, L.-C. Lam, and T.-c. Chiueh, "Bird: Binary interpretation using runtime disassembly," in *Proceedings of the International Symposium on Code Generation and Optimization*, CGO '06, pp. 358–370, IEEE Computer Society, 2006.

[19] P. Team, "Address space layout ramdomization." pax.grsecurity.net/docs/aslr.txt, 2001.

[20] E. G. Barrantes, D. H. Ackley, T. S. Palmer, D. Stefanovic, and D. D. Zovi, "Randomized instruction set emulation to disrupt binary code injection attacks," in *Proceedings of the 10th ACM conference on Computer and communications security*, pp. 281–289, 2003.

[21] R. Lyons and W. Vanderkulk, "The use of triple-modular redundancy to improve computer reliability," *IBM Journal of Research and Development*, vol. 6, no. 2, pp. 200–209, 1962.

[22] A. J. Mohammad, D. Hutchison, and J. P. G. Sterbenz, "Towards quantifying metrics for resilient and survivable networks," in *Proceedings of the 14th IEEE International Conference on Network Protocols (ICNP)*, pp. 17–18, November 2006.

[23] J. P. Sterbenz, E. K. Çetinkaya, M. A. Hameed, A. Jabbar, and J. P. Rohrer, "Modelling and analysis of network resilience (invited paper)," in *Proceedings of the Third IEEE International Conference on Communication Systems and Networks (COMSNETS)*, (Bangalore), pp. 1–10, January 2011.

[24] J. P. Sterbenz, E. K. Çetinkaya, M. A. Hameed, A. Jabbar, Q. Shi, and J. P. Rohrer, "Evaluation of Network Resilience, Survivability, and Disruption Tolerance: Analysis, Topology Generation, Simulation, and Experimentation (invited paper)," *Springer Telecommunication Systems*, pp. 1–32, 2011. Published online: 7 December 2011.

[25] A. Jabbar, H. Narra, and J. P. G. Sterbenz, "An approach to quantifying resilience in mobile ad hoc networks," in *Proceedings of the 8th IEEE International Workshop on the Design of Reliable Communication Networks (DRCN)*, (Krakow, Poland), pp. 140–147, October 2011.

[26] E. K. Çetinkaya, D. Broyles, A. Dandekar, S. Srinivasan, and J. P. G. Sterbenz, "A comprehensive framework to simulate network attacks and challenges," in *Proceedings of the 2nd IEEE/IFIP International Workshop on Reliable Networks Design and Modeling (RNDM)*, (Moscow), pp. 538–544, October 2010.

[27] J. P. G. Sterbenz, D. Medhi, B. Ramamurthy, C. Scoglio, D. Hutchison, B. Plattner, T. Anjali, A. Scott, C. Buffington, G. E. Monaco, D. Gruenbacher, R. McMullen, J. P. Rohrer, J. Sherrell, P. Angu, R. Cherukuri, H. Qian, and N. Tare, "The Great plains Environment for Network Innovation (GpENI): A programmable testbed for future internet architecture research," in *Proceedings of the 6th International Conference on Testbeds and Research Infrastructures for the Development of Networks & Communities (TridentCom)*, (Berlin, Germany), pp. 428–441, May 2010.

[28] J. P. G. Sterbenz, D. Medhi, G. Monaco, B. Ramamurthy, C. Scoglio, B.-Y. Choi, J. B. Evans, D. Gruenbacher, R. Hui, W. Kaplow, G. Minden, and J. Verrant, "Gpeni: Great plains environment for network innovation." http://wiki.ittc.ku.edu/gpeni, November 2009.