Quantifying Cyber-Resilience Against Resource-Exhaustion Attacks

Glenn A. Fink, Richard L. Griswold, Zachary W. Beech
Pacific Northwest National Laboratory
Richland, WA

***Abstract*—Resilience in the information sciences is notoriously difficult to define much less to measure. But in mechanical engineering, the resilience of a substance is mathematically well-defined as an area under the stress-strain curve. We combined inspiration from mechanics of materials and axioms from queuing theory in an attempt to define resilience precisely for information systems. We first examine the meaning of resilience in linguistic and engineering terms and then translate these definitions to information sciences. As a general assessment of our approach's fitness, we quantify how resilience may be measured in a simple queuing system. By using a very simple model we allow clear application of established theory while being flexible enough to apply to many other engineering contexts in information science and cyber security. We tested our definitions of resilience via simulation and analysis of networked queuing systems. We conclude with a discussion of the results and make recommendations for future work.**

Keywords*—*Resilience, material science, information science, stress, strain, cyber systems

I. Introduction

This paper explores the meaning of *resilience* for computational systems. We apply an understanding of resilience used in mechanical systems in hopes of discovering new ways to measure resiliency in information systems.

We begin with a discussion of the semantics of the word "resilience." We identify several salient characteristics of resilience as it is defined in the material sciences and attempt to translate these attributes for use in information systems. Next, we propose a mathematical definition of resilience in information space based on this direct translation.

Given our basic quantification of resilience we defined a simulation environment where our definitions may be tested. We conducted simulation studies and discuss the effectiveness of the definitions. We conclude with recommendations for our expected follow-on work.

II. Background

Resilience may be defined in physical terms as, "the capability of a strained *body* to recover its *size* and *shape* after *deformation* caused especially by compressive *stress*"[1], or, "the ability of a *substance* or *object* to spring back into *shape*; *elasticity*"[2], or as, "the ability of a *material* to absorb *energy* when it is *deformed elastically*, and release that *energy* upon *unloading*"[3](emphasis added).

[1]Merriam-Webster's online dictionary
[2]The New Oxford American Dictionary
[3]Wikipedia

In information sciences, Rey *et al.* distinguish among *robustness*, *resilience*, and *dependability*: "If the quality of service is temporarily violated due to the magnitude of the perturbation, the system is not robust, but it is considered resilient, as long as it is able to recover and resume operations within acceptable levels of service. The system is said to be dependable if most of the time it provides an acceptable service" [1]. Thus robustness has to do with the magnitude of the tolerable degree of disturbance while resilience deals with the elastic recovery of the system.

Most authors in information sciences define resilience purely in terms of the availability of the system concerned [2]–[4]. Others insist that beyond availability, resiliency includes ability to withstand threats of an unexpected and malicious nature and the classic protection qualities of confidentiality and integrity [5], [6]. Some explicitly include defense and recovery with respect to cyber attacks in their definitions [7] and prototypes [8]. Arguably, one could include these protection properties under the broad rubric of "operational normalcy," but we believe making security concerns explicit is fundamental.

It is perhaps self-evident that many natural systems can be described as resilient and that nature is an excellent storehouse of resilient designs that can be emulated. Brand and Jax [9] call out two distinct meanings of resilience, (1) **engineering resilience**, the *time* required to return to a nearby equilibrium point following a disturbance, and (2) **ecological resilience**, the *amount of disturbance* a system can absorb before changing to a new equilibrium point, possibly distant from the original, "controlled by a different set of variables and characterized by a different structure."

Because most engineered systems are incapable of completely reconfiguring themselves in the face of a severe disturbance, we define resilience as returning to a *nearby* equilibrium point but do not quantify it in terms of time. We realize that a goal of computational system engineering is to make systems that are resilient by the second definition, and initial attempts to achieve this are underway [10]. For now, however, we will be satisfied to lay the groundwork for a quantifiable definition of engineering resilience in the hopes that our definitions may be extended to measure the success of a system claiming something like ecosystem resilience should one be designed.

Clearly, under some set of circumstances, all systems will fail. Resiliency in the engineering sense is not about preventing failure but in managing *how* the system behaves during failure and recovery. We must be careful not to sloppily redefine resilience as goodness. In engineering, "goodness" must always be quantified in terms of relative cost, and the

978-1-4799-4187-2/14/$31.00 ©2014 IEEE

cost of resilience must be considered relative to the cost of the kinds of failure anticipated.

If the cost of unmanaged failure and recovery is low enough, a brittle system may be an appropriate solution. Resilience may be bad if employing it costs more in lost system performance than the cost of occasional system failure and recovery. For instance, rebooting a personal computer or restarting a hung application is a common, if unpleasant occurrence. The cost is relatively low and the safety risk is zero. However, restarting a nuclear reactor or a space life-support system has much greater cost and the risks can often be measured in lives. In this case, resilience is required.

While these remarks have clarified the value of resilience and its meaning, this paper is not primarily about assessing whether resilience is good or bad in any particular engineering context. Instead, we concentrate on quantifying how resilience may be measured with respect to a phenomenon like the dropping of packets or jobs in a queuing system. We expect this simple application will allow clear application of established theory while being flexible enough to apply to many other engineering contexts in information science and cyber security.

III. Applying the concept of resilience

To apply definitions of resilience to various domains we must find analogs for:

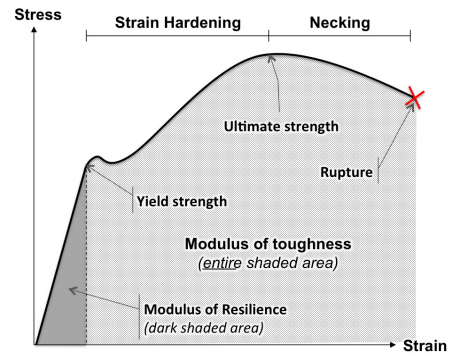
- The subject that bears the load (*e.g.*, the body, substance or object)
- The measured attributes of the subject (size and shape)
- The stress or per-unit-area force loaded upon the subject
- The result of this stress, (deformation or *strain*)
- The behavior of the object after deformation (elasticity or energy unloading)

A. Resilience in materials

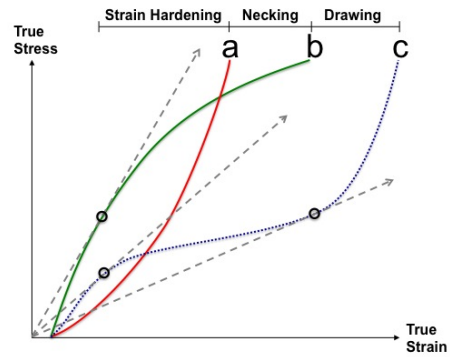
In material systems resilience is defined in terms of the relationship between stresses exerted on an object and the strains (or deformation) that object exhibits. Stress is defined as average load (force), P, per unit area, A, where force is the product of mass and acceleration, and area, dimensionally, is squared length. In material systems, strain is a measure of physical displacement (length) at a given stress level.

In material science, materials deform elastically up to a certain amount of stress after which permanent distortion occurs. The strain point where this distortion occurs is called the "yield point." The *modulus of resilience* of a material is the maximum energy that can be absorbed per unit volume without creating a permanent distortion and may be calculated as the area under the stress-strain curve (Fig. 1a) from zero stress up to the yield point [11, p. 698]. Thus, in materials science, resilience is a well-defined phenomenon.

Not all materials have a yield point, however. Some materials rupture prior to yielding, others form a strain hardened "neck" after yielding, and still others both form the neck and "draw" more material into the neck. The resulting nonlinear relationship between stress and strain characterizes the material's toughness and resilience and can be analyzed via the Considère construction (Fig. 1b) [12].



(a) Engineering stress-strain curve for a ductile material



(b) Considère construction for true stress-strain curves

Fig. 1: Types of stress-strain curves.

The first type of materials are brittle (Fig. 1b, curve a), and they fracture at the yield point. No secant line can be drawn from the origin to form a tangent with this kind of curve. The second type, the ductile materials, (Fig. 1b, curve b) thin to a neck after their yield point. Only the neck absorbs the remaining strain, and the material will break there. A single secant line can form a tangent with the curve at the yield point. The third type of material is the elastomers (Fig. 1b, curve c), which also form a neck at the yield point. But additional stress causes elastomers to undergo a structural transformation. Strain-hardening causes material from the shoulders adjacent to the neck to flow into the neck region. As stress increases, these materials exhibit a second round of strain hardening marked by the second secant-line tangent point.

The shape of the stress-strain curve contrasts the failure modes of brittle, ductile, and elastic materials. Brittle materials may be very strong, but they fail suddenly, often without warning. Ductile materials deform before failure indicating overloading and providing some warning [11, p. 32]. Elastic materials are highly resilient, and they may or may not deform before they fail, but the strain is obvious when stress is applied. Brittleness is not pejorative—concrete is brittle, but it is used ubiquitously. Sudden failure may be acceptable if in trade, one can eek out a bit more pre-failure performance.

The *modulus of toughness* [11, p. 697] is equal to the area under the entire stress-strain diagram to the point of rupture (Figure 1a). It represents the energy per unit volume required

to cause the material to fracture. A material's toughness is its ability to absorb energy. Since resilience is a subset of toughness in material science there is no conflict between them. All other things held constant, if the modulus of resilience is larger, the modulus of toughness would not decrease.

B. Resilience definitions for information systems

An argument could be advanced that material systems are much less complex than information systems and direct comparison is not reasonable. While this is likely true for simple, man-made materials, biological materials are far more complex and contain intricate hierarchies of proteins, microfibrils, and other structures laid out with all the complexity of a program. These materials exhibit highly nonlinear stress-strain reactions [13], [14] that are only poorly understood at this time. However, the same stress-strain analysis can shed light on these complex behaviors.

Even though material stress can dimensionally be reduced to just mass, distance, and time, translating the physical units to equivalent units in information space is nontrivial, making resilience much more difficult to define in this domain. For instance, area (distance) is not a well-defined dimension in information space. There is no concept of absolute direction and thus no velocity vectors. Although there are locations in cyber space, without the concept of direction, relative location is not well defined in any meaningful sense. Similarly, material strain is a measure of physical displacement (length) at a given stress level, but length is not well-defined in cyber space.

To enable clear discussion of the meaning of resilience in information systems, we adapt terms from material mechanics [11] using ideas from queuing theory [15, pp. 505–626]:

a) System: A collection of queues and their attendant consumers that process submitted jobs. The queues may be interdependent and the entire system is the subject of the stress.

b) Load: The amount of work the system is given to perform each unit of time. Work includes reception and queuing of jobs, execution of jobs and delivery of output.

c) Capacity: The sustained load that can be accomplished per unit time. Capacity may aggregate resources such as number of queues, number of processors, memory, *etc*.

d) Stress: The load divided by the system's capacity. In a network of parallel queues, stress would be the load divided by the number of queues.[4] In material science physical deformation causes a difference between "true" and "engineering" stress/strain, but in information systems as long as there are no nonlinear performance changes caused by stress, there is no difference.

e) Strain: Alteration of the system's performance caused by the work it must perform. In material systems strain measures the physical change in length of a sample at a given stress level compared its resting dimensions. Naïvely, in cyber systems, displacement might be measured as changes in the response time at a given stress level. However, measuring displacement in terms of time rather than in units analogous to some length metric causes lack of orthogonality problems that make the candidate metric difficult to combine with others.

Strain may be related to queue length or other performance indicators. In a simple queuing system, the strain on an individual queue is the number of jobs in the queue at a given load, minus the queue length when the system is under zero load, divided by the capacity of the queue (number of buffers). Since zero load produces zero queue length, strain is simply the sum of the current queue lengths divided by the sum of the queue capacities.

f) Resilience: The workload the system can accept up to the point of a non-recoverable fault. Please note that the concept of post-failure recovery is not found in material science, although it is found in biology. Unless the system crashes, cyber systems are often resilient up to the crash point– they return to their original capacity once the excess load is removed. Failure points caused by things like deadlock or memory leaks could be modeled by introducing an arbitrary failure point that happens stochastically after some amount of loading or uptime.

g) Toughness: The closest analogue to toughness as defined in material engineering may be what information scientists call "robustness." We arrive at that conclusion because toughness has to do with the strength at the rupture point of the material. Similarly, robustness is the ultimate ability of the system to accomplish workload even beyond the point where fatal faults have happened and eventual failure is certain. By this definition, the robustness of a system will never be less than its resilience.

Because resilient systems emphasize fault tolerance it may seem they must trade in some robustness to gain reliability, but hybrid approaches are possible and the simulation results we present later illustrate this. The system characteristics that are actually in conflict are brittleness and ductility. We do not redefine these characteristics for information systems quantitatively, but, given the above, one might expect robust to mean, "strong but brittle" and resilient to mean "possibly weaker but ductile."

A robust system should maintain optimal performance even when it is very close to failure. When it fails, it may fail suddenly *if it is also brittle*. In contrast, one would expect resilient systems to be more characterized by ductility. A resilient system may gradually deform (perform worse) with increased stress, but it should be able to absorb much stress without failure and it should be able to recover quickly once it does fail. Hybrid systems are possible and often desirable. Interestingly, if a system is resilient all the way to the point of a failure (e.g., there is no yield point where the system continues to operate but will fail) the modulus of resilience equals the modulus of toughness, and resilience equals robustness.

C. Cyber resilience defined quantitatively

For this work we consider a simple system with a set of job generators (the clients) and a a set of servers that process the jobs. Servers are modeled as M/M/1 queues.[5] Although this model is very limited, it is also flexible enough to adapt to

[4] We consider only the number of queues *not* the total buffer positions because the definition of strain takes this capacity into account.

[5] An M/M/1 queue has exponential arrival and service rates and is served by a single processor.

many applications via added behavioral rules and constraints. In this section we will analyze the dimensions of the problem previously laid out and apply them to information space.

a) Load: Two kinds of load must be considered separately even for a simple queuing system: (i) the number of jobs arriving at a given time, L_j, and (ii) the amount of resources required to process the jobs at that time, L_p.

In queuing theory, the queue length is a function of the mean arrival rate (λ) and the mean job service rate (μ). Traffic intensity (ρ) is the ratio of these values ($\rho = \frac{\lambda}{\mu}$). Thus, if a queuing system receives 150 jobs per second and processes only 100 per second, the traffic intensity is 1.5. This makes L_j simply $1/\lambda$.

L_p is intuitively the load due to the *size* of the jobs. In information systems this can only be reckoned semantically and may best be measured by the amount of computing power, memory, storage, and other resources that will be required to process the information. The number of bits required to represent a job is really no indication of its impact on the computing resources. In general we do not know size *a priori*. For simplicity we reduce all the dimensions of size down to the mean number of resources required by a job.

Job resource requirements are only important when the job is executing, and they may grow and shrink over time. If the number of resources required by executing jobs exceeds the available resources, we risk a system crash.

We hide the multitude of details by modeling the processing size of a job as a Poisson random variable, S whose value is known only after the job is completely processed. Then the processing load on a system is, $L_p = \sum_{j=1}^{k} S_j$ at a given time, where k is the number of jobs currently executing. Unfortunately, because the units are so different, L_j and L_p must be analyzed separately. Our simulation implements calculation of L_p, but we assume it to be negligible in our stress-strain analysis.

b) Capacity: Internally, each M/M/1 queue has a job capacity C_j equal to the number of queue buffer positions plus the number of buffers for executing jobs in the processor (typically one). But from the outside, the job capacity of the overall system is q, the number of queues available (regardless of size).

The processing capacity, C_p, is a static number of resources available at a time. For simplicity, we consider the processing capacity to be somewhat equivalent to the available CPU power making CPU utilization an apt measure of the stress on the system. We save the study of utilization for future work.

c) Stress: We define stress as the load over the capacity, and since we are talking about dynamic systems, we must measure it per unit time. We adopt the material science notation σ for stress and define engineering stress, σ_e, as a polynomial over the sources of stress (job arrivals and resource usage) as shown in Eqn. 1.

$$\sigma_e = a(L_j/q)^b + c(L_p/C_p)^d \tag{1}$$

$$\sigma_t = \sigma_e(1 - \frac{\text{jobs}_{dropped}}{\text{jobs}_{total}}) \tag{2}$$

For some constants a, b, c, and d. To simplify our analysis we set all the constants to one and set $L_p = 0$, making $\sigma_e = L_j/q$. "True" stress could take into account the proportion of dropped jobs produced by a given regimen (Eqn. 2). This assumes that dropping jobs is undesirable; however, dropping jobs may be a resilience strategy to avoid a crash. So the appropriateness of true stress as a metric depends on the needs of the designers.

d) Strain: For a simple queuing system, the queue length, n, is the natural measure of strain on the system and it is unitless, avoiding orthogonality worries. Physical objects have a nonzero resting length, but the *resting* queue length of an information system should be zero. Thus, any utilization of the system induces strain.

Strain must be normalized both in material and information systems to keep measurements comparable. In material systems, standard strain, ϵ is deformation distance divided by the length of the sample. In information systems, the same approach allows us to say that $\epsilon = n/C_j$.

e) Resilience: Resilience may now be simply defined as the area under the stress-strain curve of a given information system. What remains is to show what this curve looks like and determine whether one system that is arguably more resilient than another actually shows a higher resilience using this method.

Since resilience includes recovery effort, lost processing capability during recovery time may be added to measure the system's elasticity. This may be measured by summing the instantaneous resilience metric over time and subtracting out the portion that would be missed during downtime. However, we do not include recovery time in this analysis.

IV. Analyzing cyber resilience

For over half a century, queuing theory has been used to make theoretical predictions about the operation of real information systems. Queues provide a highly flexible modeling approach and are well studied. We may assign artificial behaviors and constraints to queuing models and network them in a variety of interdependencies to model the behavior of even complex information systems.

For instance, to model resource exhaustion, we may make a queue accept jobs more slowly as it becomes more full. A system crash may be illustrated by making a queue drop all current jobs and not accept new ones for a period of time. To emulate system errors, we may selectively drop jobs, or process them and not return them. Complex systems may be represented by having queues send jobs to one another in arbitrarily complex networks.

Under the assumption that queuing models provide an adequately flexible primitive for real systems, we next discuss our queue-resilience simulation model and analyze some of its results to determine whether our proposed measures of resilience hold true.

A. Simulating cyber resilience

To simulate resilience behavior of various system configurations, we created a discrete-event simulation in NetLogo [16]

representing a set of clients and servers in a job-processing system arranged in a fully connected bipartite mesh. Each client may send jobs to any server and, depending on the policy, the servers may send jobs to one another. Every time step each client generates a number of jobs determined by a Poisson random variable and sends them to various servers for processing. The servers form an open network of M/M/1 queues [15, ch. 32].

If a job arrives at a server whose queue is full, depending on the queuing discipline, the queue may drop the job or forward it to another server that is less full. The policy selected will affect the resilience of the overall system. When a job is complete, the processing server returns the job to the client with the run statistics collected for that job, and the client records them. Jobs that successfully return to the client are tallied as complete, and the system keeps statistics on the fate of all jobs whether complete or dropped.

As discussed previously, the size of each job is *a priori* unknowable because of the halting problem. In the simulation we thus use a Poisson random variable with a user-selectable mean whose value can be randomly generated at arrival at the server—no job takes less than zero time to complete and any job may never terminate, although we consider the probability of a non-terminating job to be remote normally. To simulate malicious jobs it would be simple to create jobs with very high resource demands that would cause the server to hang or crash. By default, we assume the same mean size for all jobs.

Similarly, we define the capacity or processing power of each server in terms of the job size it can complete in unit time. The server's capacity is subtracted from the remaining duration of the jobs it has to work on each time step. If a server finishes executing a job and has remaining capacity it immediately begins executing another job.

Additionally, every job has a resource usage that cannot be known in advance. The resource usage represents in a single value all the potential system resources a job might need (*e.g.*, disk space, stack/heap allocations, shared resources) as it executes. Because jobs are dynamic, the resource usage may change over the job's lifetime. We assume non-decreasing resource usage of four types: unit-sized non-growing, exponential-sized non-growing, linearly growing by unit size, and exponentially growing. This nuance allows us to investigate the effects of resource exhaustion in a limited sense.

We have enabled two policies for handling arriving jobs: a dropping policy and a forwarding policy. The dropping choices are either to simply drop arriving jobs when the queue is full or to have an exponentially increasing chance of dropping them as the queue fills. Given n, the current count of jobs in the queue, and b, is the maximum the queue can hold, gives the dropping probability for each policy:

$$p(\text{drop}) = \begin{cases} \text{When full:} & = \begin{cases} n < b : 0 \\ n \geq b : 1 \end{cases} \\ \text{Exp. prob.:} & = p(e^{n/b} > r); r \in [0, e] \end{cases} \tag{3}$$

The forwarding policy is simply a boolean value, which, if true causes queues to forward jobs to another randomly selected queue with fewer jobs. If no such queue is found the job is dropped. This enables emulation of load sharing. Because the actual size of a job is not knowable in advance, queues may only make forwarding or dropping decisions based on how many jobs are currently queued, not on the size of jobs queued or in process. A busy queue cannot predict when it will be free other than by its queue length.

Forwarding and dropping may be combined to yield four policies: *Control*: drop when full, *Expdrop*: drop with exponentially increasing probability, *Forward*: forward when full, and *Expfwd*: forward with exponentially increasing probability. We refer to these strategies by their labels hereafter.

B. Experiment

The question this paper seeks to answer is whether resilience can be characterized and computed for information systems just as it can for material systems, and whether there is more than one type of resilience for these systems. Thus we must derive usable measures of stress and strain and determine whether these metrics are actually useful in characterizing and comparing systems and resilience strategies.

Defining resilience with respect to (w.r.t.) queues implies an overfull queue is a failure state. Resource exhaustion attacks like distributed denial-of-service (DDoS) attacks against a web server can be modeled in this way. The individual jobs are usually requests for information that are not as important as the availability of the server itself and can be easily reissued.

Defining resilience w.r.t. jobs implies that the jobs themselves may be critical and that to lose one is worse than almost anything shy of bringing the whole system down. This is the approach taken by the Linux Out-of-Memory (OOM) Killer [17]. The OOM Killer tries to selectively sacrifice noncritical jobs when the system is running critically low on its free memory. It scores each job by its likelihood of being something that the user really cares about and is more likely to kill jobs that will (1) release a lot of memory, (2) are lower priority, and (3) are not hardware controllers.

Given our definition of resilience as the area under the stress-strain curve, our hypothesis was that *Expdrop* would be the most resilient policy for keeping the queue from overloading. This is because it preemptively begins dropping jobs as the queue fills. Similarly, we expected *Forward* to be a more resilient policy than *Control* w.r.t. jobs because if one queue should fail, the job is not lost, it would simply be shunted to another queue. We expected that *Expfwd* should be more resilient than *Forward* w.r.t. jobs for the same reasons that *Expdrop* should be more resilient than *Control* w.r.t. queues.

C. Simulation results

We ran our discrete-event simulation with five clients and five servers under all four policies at varying traffic intensities[6] from $\rho = 0.5$ to 6.0 for 20,000 time steps, ignoring the first 700 time steps to allow the queues to reach a stable state. Queue length for all queues was 30 plus one job in execution. At $\rho = 2.0$ 20,000 steps produces about 750,000 jobs per client scenario and an average of 3,740,000 for all five clients.

[6]Recall that traffic intensity, ρ, is the ratio of the arrival rate over the job processing rate.

We then calculated means of the following at each ρ level:

1) job arrival rates,
2) job completion rates (counting only jobs that were not dropped),
3) number of jobs in the queue at the time of job arrival,
4) number of jobs dropped by the queue at job arrival,
5) system strain level due to job arrivals,
6) system true stress level due to job arrivals, and
7) system engineering stress level due to job arrivals.

Although we gathered observations on response time, waiting time, throughput, etc. we did not include them in our analysis. We also did not conduct any analysis of stress or strain due to job processing. These we leave for future work. Fig. 2 shows the plots of the stress-strain values for each traffic intensity and lists the area under the curve (AUC), or modulus of resilience for each policy.

We note that the curves in Fig. 2a exhibit a smooth, non-decreasing 'S' shape without a linear-elastic section, yield point, or ultimate strength. Early in this research, we were dismayed that our stress-strain curves for queues showed none of these. To remedy this we attempted to identify functional "yield" and "rupture" points at the knees of the curves, but this overfitting turned out to be unnecessary as many materials do not have a yield point either because they are nonlinearly elastic (as seen in Fig. 1b, curve c) or because they are exceptionally stiff and brittle and break before they yield. The latter we would define more as robustness than resilience.

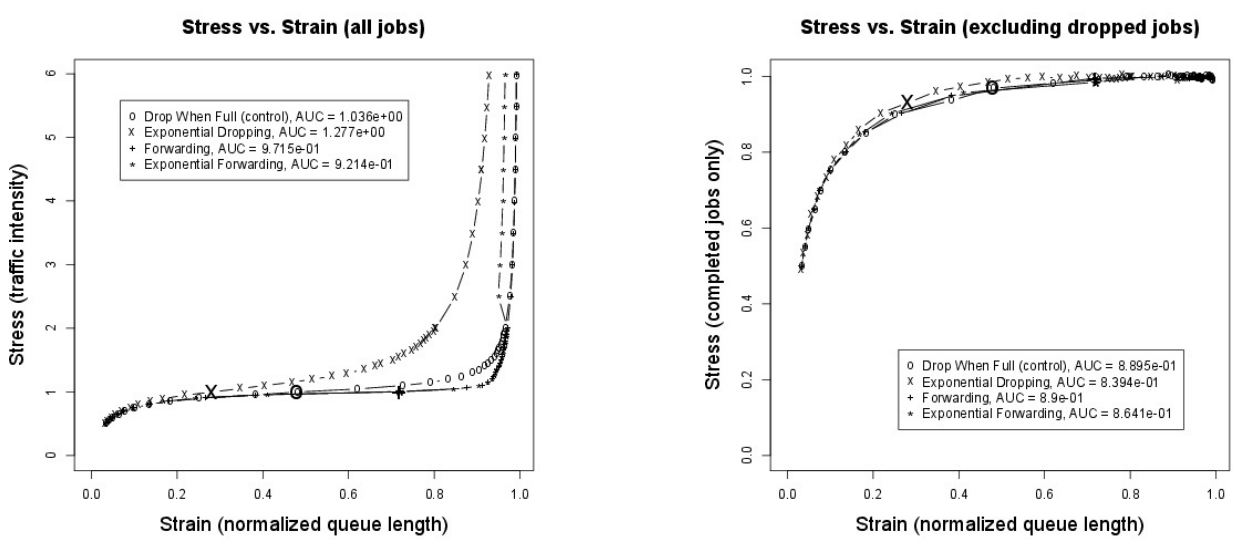
The shape of our curves closely resembles stress/strain curves for elastomer materials that typically are very resilient and tough. This means that the queue has a lot of extra capacity that is not used until the stress increases enough to mostly fill the queue. We expected a more linear queue-filling response to traffic intensities. We also note that although the values of ρ from 0.5 to 2.0 are evenly spaced at 0.05 intervals, in all the curves, there is a marked sparsity of points near $\rho = 1.0$. The queue length undergoes a relatively rapid transition from mostly empty to overfull at around this point.

The sparsity of points at $\rho > 2.0$ arises because we used an interval of 0.5 at high levels to reduce the time required by the simulations. All the policies would eventually asymptotically reach a strain of 1.0, but choosing a maximum of $\rho = 6.0$ allowed us to compare the curves acceptably fairly. Since these curves do not have a yield point, the entire area under each curve is that policy's modulus of resilience. *Expdrop* had the most resilience even though we were unable to drive that curve to its asymptotic maximum. Interestingly, the *Forward* and *Expfwd* policies performed worse than the *Control* strategy.

Fig. 2b compares the strategies using our measure of "true" stress, where stress is only counted for jobs that complete. Interestingly, all the strategies appear to perform similarly, and all only reach a true stress level of 1.0 even though ρ reached 6.0 in all cases. This makes sense because above $\rho = 1.0$ jobs must be regularly dropped since the queue will be full. By this metric, the modulus of resilience is practically indistinguishable regardless of the strategy.

Our measure of resilience rewards algorithms for keeping the queue length short under increasing traffic intensities. We would expect the most resilient methods would also have the worst utilization of the total queue buffer space. Similarly, if we care about resilience with respect to jobs, not queues, then a resilience strategy that depended on dropping jobs would be bad even if it risks crashing the whole system. From this viewpoint, the strategy that (on average) drops the fewest jobs is the most resilient with respect to job survival.

If we plot the queue length vs. the number of dropped jobs at various values of traffic intensity (ρ) we obtain curves



(a) Engineering stress vs. strain

(b) "True" stress vs. strain

Fig. 2: Multiple queue simulations comparing engineering and "true" stress-strain at varying traffic intensities. The enlarged symbols mark the points where $\rho = 1.0$ on each line.

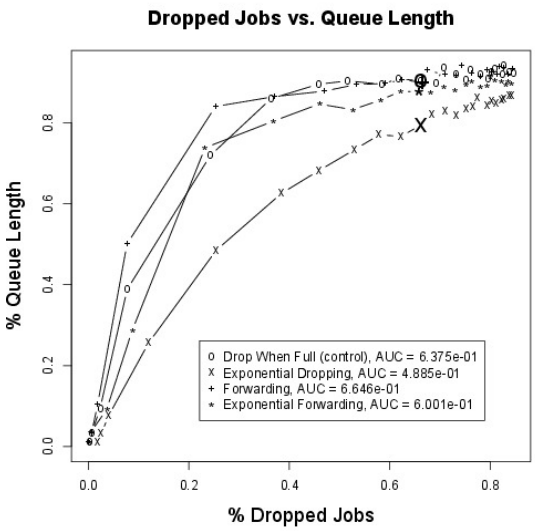


Fig. 3: Comparison of queue length vs. dropped jobs. The enlarged symbols mark the points where $\rho = 1.0$.

like those shown in Fig. 3. If area under the curve gives an indication of resiliency, *Expdrop* is the least resilient and *Forward* the most with respect to jobs. The results are nearly the inverse of resilience w.r.t. queues. Resilience results are summarized in Table I and discussed in section IV-D.

TABLE I: Moduli of Resiliency for each policy with respect to queues and jobs. Ranks are given after each modulus.

	Modulus of Resilience		
	w.r.t. queues	queues excl. drops	w.r.t. jobs
Control	1.036 #2	0.889 #2	0.6375 #2
Expdrop	1.277 #1	0.839 #4	0.4885 #4
Forward	0.921 #4	0.864 #3	0.6646 #1
Expfwd	0.972 #3	0.890 #1	0.6001 #3

D. Discussion

Considering first the results shown in Fig. 2a we see that *Forward* and *Expfwd* policies have flatter growth along the majority of the queue lengths and much sharper upturns toward the end than the other two. We interpret this to mean that these policies avoid dropping jobs as much as possible until all the queues are full, then they all dump the extra jobs together. At first glance we would not consider this capacitative behavior resilient, it could be considered highly robust but brittle—it withstands failure until all queues fail, then it fails massively.

The forwarding policies may recover more quickly because they dump jobs in large batches, but recovery time cannot be observed directly in the data. *Expfwd* ameliorates the behavior of *Forward* by forwarding jobs earlier and with increasing likelihood. This is borne out by the slower asymptotic ascent at the end of the curve.

Resource-exhaustion attacks can crash systems using forwarding strategies, but if there is enough redundant provisioning the job-dumping behavior may allow them to quickly recover and work at peak utilization again. However, the forwarding approaches cheat a bit because the time it takes for a job to be rerouted actually functions as additional queue length, giving them added resilience up to the point of total failure. Then when all the queues are full all the transmitted jobs get dropped until more processing space is available.

In contrast to the forwarding strategies, the *Expdrop* policy has a smooth curve showing clearly how more jobs are dropped as the average queue length grows. At a given traffic intensity *Expdrop* will never drop as many jobs as the forwarding policies and the queues almost never become full. Thus, no recovery by dumping masses of jobs is necessary.

The *Expdrop* policy has the most regular spacing of the ρ points of all the curves. Thus, we expect it to fail more gently than control and under greater stress levels. *Expdrop* shows resilience because it never reaches the strain levels of the other three policies at the traffic intensities tested. A strategy like this may reduce the effects of resource-exhaustion attacks, but throughput and utilization may suffer, although we did not collect data on those quantities for this experiment.

The *Control* policy takes the middle ground, appearing to get neither maximal utilization out of the queues nor avoiding resource exhaustion. But this does not exclude it from being considered as a potential solution when selecting an algorithm. Interestingly, *Control* is consistently the second-best policy. Thus, one might consider *Control* the most resilient given its consistency in all conditions.

Considering now the results from Fig. 2b we see that the strategies are nearly identical when stress is only counted from jobs that are not dropped. Since dropping jobs is the primary means of enacting resilience w.r.t. queues, it is not surprising that penalizing an approach for dropping jobs would erase the advantage of doing so. But the near identical curves in Fig. 2b do show that when dropping is not taken into account, the approaches are nearly identical. The only remaining discernible difference is that the forwarding strategies fill their queues more quickly at lower traffic intensities.

The results shown in Fig. 3 show that *Expdrop* is clearly the least resilient policy w.r.t. jobs. This is because it preemptively drops jobs long before the queue is nearly full. *Forward* is clearly more resilient than the others for $\rho \leq 1.0$ but less distinctly the winner for high values of ρ. Contrary to our expectations, *Expfwd* was less resilient w.r.t. jobs than *Forward*. This seems to occur because the preemptive forwarding puts many more jobs in the netherworld between queues than *Forward*. Thus, when the queues do ultimately fail, many more jobs are dropped.

V. Conclusion

The metrics of resilience we selected work as desired, and they highlighted features of the policies that were previously unnoticed. Failure and recovery modes became clearer under the light of this analysis and we were able to see quantifiable differences in policy behaviors. Our analysis also explains why resilience and robustness are different but so often confused.

Queuing models may be modified to represent most any system behavior. Thus, our approach will enable us to test resilience models against most any kind of system with measurable and interpretable results. To test a given information system for resilience, one must be able to measure the strain

induced at varying stress levels. Then the resilience of various strategies can be compared directly.

This analysis can also be done indirectly by analyzing a queuing model abstracted from the real system. Using the metrics we have described, abstract models with sufficient fidelity to the real systems can predict performance characteristics of those systems quantitatively.

A. Future Work

Much more work is required to bring our approach to fruition. We must understand whether other concepts such as yield points and ultimate strengths from materials science might translate to information science. We must investigate the behavior of processing stress and strain to see if it is different from that of queuing effects seen in job-arrival stress and strain. An indication of the effects of recovery time is a worthy follow-on research topic. Then we must find a way to combine these multiple dimensions of stress and strain to establish a holistic view of information-system resilience.

We must also seek metrics that will tell us what kind of resilience is important in a policy or algorithm for a particular use. It is interesting but insufficient to simply differentiate algorithms quantitatively. More direct mapping of the theoretical queuing models here to real-world systems must be made. Our method is sound, but we do not fully know which kind of resilience is "good." Further research may enlighten this.

Our simulation focused on two types of resilience: w.r.t. queue and w.r.t. jobs. These translate roughly into resilience of availability and resilience of integrity. We did not directly address confidentiality as Bishop, *et al.* insist resilience must [5]. However, availability applies not only to the system, but also to its cyber defenses. Through further research, we hope to show that availability resilience of the cyber defenses results in the inability of an adversary to disable or deny the defenses and thus can result in improved resiliency of confidentiality, integrity, and availability of the entire system.

B. Contributions

We have defined resilience in a quantifiable way as the area under a stress-strain curve, and the definition itself adds clarity to the discussion of the nature of resilience. Obviously, material science theories are not directly translatable into information spaces because of the very different natures of matter and information. However, by framing the discussion this way, we have begun to define resilience quantifiably for information systems with metrics that support our definitions.

Quantifying resilience of simple systems is the first step towards quantifying resilience of complex systems. We believe that just as queuing models have been very useful in engineering real-world systems, so resilience studies of this nature hold great promise. Queuing has, after all, been studied for about half a century, and many applications have arisen from it. Mechanical engineering has grown over many centuries to the state it is now. We hope this new way of thinking about resilience in information systems will be another success from synthesizing learning from established fields.

Acknowledgement

This work was supported by the Asymmetric Resilient Cybersecurity Initiative at the Pacific Northwest National Laboratory (PNNL) and corresponds to report number PNNL-SA-102246. PNNL is a multi-program national laboratory operated by Battelle Memorial Institute for the US Department of Energy under DE-AC06-76RLO 1830.

References

[1] G. Zambrano Rey, M. Carvalho, and D. Trentesaux, "Cooperation models between humans and artificial self-organizing systems: Motivations, issues and perspectives," in *Resilient Control Systems (ISRCS), 2013 6th International Symposium on*, Aug 2013, pp. 156–161.

[2] J.-C. Laprie, "From dependability to resilience," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN) (Fast Abstracts)*, 2008.

[3] C. Rieger and Q. Zhu, "A hierarchical multi-agent dynamical system architecture for resilient control systems," in *Resilient Control Systems (ISRCS), 2013 6th International Symposium on*, Aug 2013, pp. 6–12.

[4] A. Melin, E. Ferragut, J. Laska, D. Fugate, and R. Kisner, "A mathematical framework for the analysis of cyber-resilient control systems," in *Resilient Control Systems (ISRCS), 2013 6th International Symposium on*, Aug 2013, pp. 13–18.

[5] M. Bishop, M. Carvalho, R. Ford, and L. M. Mayron, "Resilience is more than availability," in *Proceedings of the 2011 Workshop on New Security Paradigms Workshop*, ser. NSPW '11. New York, NY, USA: ACM, 2011, pp. 95–104. [Online]. Available: http://doi.acm.org/10.1145/2073276.2073286

[6] C. Rieger, D. Gertman, and M. McQueen, "Resilient control systems: Next generation design research," in *Human System Interactions, 2009. HSI '09. 2nd Conference on*, May 2009, pp. 632–636.

[7] Y. Yuan, Q. Zhu, F. Sun, Q. Wang, and T. Basar, "Resilient control of cyber-physical systems against denial-of-service attacks," in *Resilient Control Systems (ISRCS), 2013 6th International Symposium on*, Aug 2013, pp. 54–59.

[8] D. Lam, E. Skiles, and P. Grisham, "Simulation tool for evaluation and design of resilience strategies," in *Resilient Control Systems (ISRCS), 2013 6th International Symposium on*, Aug 2013, pp. 186–191.

[9] F. S. Brand and K. Jax, "Focusing the meaning(s) of resilience: resilience as a descriptive concept and a boundary object," *Ecology and Society*, vol. 12, no. 1, 2007. [Online]. Available: http://www.ecologyandsociety.org/vol12/iss1/art23/

[10] P. Ramuhalli, M. Halappanavar, J. Coble, and M. Dixit, "Towards a theory of autonomous reconstitution of compromised cyber-systems," in *Proceedings of the 2013 Technologies for Homeland Security (HST), 2013 IEEE International Conference on*, Nov 2013, pp. 577 – 583.

[11] F. P. Beer, J. E. Russell Johnston, J. T. Dewolf, and D. F. Mazurek, *Mechanics of Materials*, 6th ed. McGraw-Hill, 2012.

[12] D. Roylance. (2001) Stress-strain curves. [Online]. Available: http://ocw.mit.edu/courses/materials-science-and-engineering/3-11-mechanics-of-materials-fall-1999/modules/ss.pdf

[13] M. J. Buehler, "Multiscale mechanics of biological and biologically inspired materials and structures," *Acta Mechanica Solida Sinica*, vol. 23, no. 6, pp. 471 – 483, 2010. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0894916611600013

[14] S. W. Cranford, A. Tarakanova, N. M. Pugno, and M. J. Buehler, "Nonlinear material behaviour of spider silk yields robust webs," *Nature*, vol. 482, no. 7383, pp. 72–76, Feb 2012. [Online]. Available: http://dx.doi.org/10.1038/nature10739

[15] R. Jain, *The Art of Computer Systems Performance Analysis Techniques for Experimental Design, Measurement, Simulation, and Modeling*. John R. Wiley and Sons, Inc, 1991.

[16] U. Wilensky, "Netlogo," http://ccl.northwestern.edu/netlogo/, Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL., Tech. Rep., 1999.

[17] R. van Riel. Linux-mm docs: the oom killer. [Online]. Available: http://linux-mm.org/OOM_Killer