

# Process Query Systems

*George Cybenko and Vincent H. Berk*

Dartmouth College

Sensors produce large streams of raw events while instrumenting environments such as computer systems, communications networks, physical spaces, and human organizations. Extracting meaningful and actionable information from these events, however, remains a challenge. Process query systems, a new algorithmic and software paradigm, offer a powerful and generic way to address event-processing challenges.

Our ability to instrument different environments has increased dramatically in recent years. Computer systems and networks now routinely include various performance monitors, firewalls, intrusion-detection systems, and application-logging agents.

Researchers can deploy sensor networks in physical environments to record acoustic, seismic, infrared, video, electromagnetic, and other types of measurements. These networks also can monitor and extensively archive communications and financial and social transactions among large communities of people and organizations.

We now have a tremendous amount of data coming at us; the question remains what to do with it.

Generally speaking, the underlying purpose of instrumenting environments is to better understand “what is going on,” formally known as *situational awareness*. In the context of computer security, situational awareness typically means knowing which monitored systems are under attack and the nature of those attacks. For a physical sensor network within a building or extending across a geographic region, this might mean being cognizant of certain objects and activities, such as a fire, people, animals, or vehicles and their location. In industrial and manufacturing systems, situational awareness means detecting infrastructure failures and diagnosing their causes.

## THE SENSOR DATA PROCESSING CHALLENGE

While the technology to instrument such environments has matured significantly, the ability to obtain situational

awareness from the resulting data streams has not kept pace. Sensed-event data is often displayed in relatively raw formats, leaving analysis and interpretation up to human operators, which ultimately is not scalable.

This has certainly been the case for computer-security-related instrumentation: Commercial software like that offered by ArcSight ([www.arcsight.com/index.htm](http://www.arcsight.com/index.htm)) can display and archive large volumes of security events collected within a corporate network, but it can conduct only a superficial analysis of the data automatically, which places a huge burden on system administrators.

Neither traditional database technology nor rule-based expert systems have proved to be up to the task of closing the gap between low-level sensor events and high-level situational awareness. Database technologies, including extensions to data-stream processing,<sup>1,2</sup> effectively store, index, and retrieve sensor reports but do not provide analysis beyond rudimentary report generation. On the other hand, decision trees and logical-rule processing inherit the well-known brittleness and scalability problems associated with expert systems. A new approach for extracting situational awareness from sensed data is therefore needed.

During the past three years, we have studied situational-awareness problems that have arisen in multiple application domains including computer security, autonomous computing, sensor networks, video tracking, and social network analysis. The variety of applications suggests that a common analytic foundation underlies many such problems. As a result, we have developed a general-

purpose framework for modeling situational awareness across multiple sensor types and environments. Using this framework, which is based on *process detection*, we have implemented a *process query system* and tested it in several applications.

### PQS MODELING FRAMEWORK

Figure 1 shows the PQS modeling framework. The fundamental premise behind PQS is that a sensed environment consists of *processes* with distinct states, dynamics, and observables (Step 1). In the case of computer security, computer attacks are the processes. Possible states of these attack processes include reconnaissance, intrusion, exploitation, and data exfiltration. Attack processes change their states over time as determined by the attack model's dynamics.

The process-oriented nature of PQS implies that an instrumented and monitored environment's static aspects are not of much interest. The changes in an environment are what we want to detect and understand. Typically, the states and dynamics of processes in an environment cannot be observed directly, but they do produce observable events and artifacts (Step 2).

This is where the sensors come in—the sensing infrastructure detects events and communicates them back to an analysis center (Step 3). The events provide evidence of the processes' states but are not identical to them. For example, it's usually impossible to know precisely about a computer attack's abstract internal state—only sensing and detecting observable artifacts of attacks such as network packets, file system changes, and system behaviors, including processor utilization, can be expected.

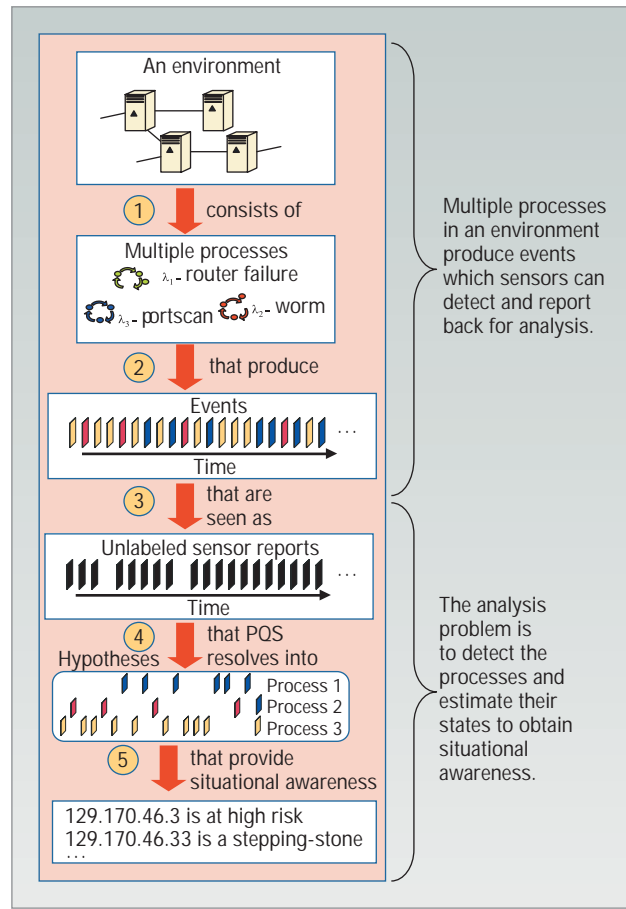
Sensor observations about the hidden, internal processes' states often consist of inconclusive and noisy evidence. Given these noisy and ambiguous sensor observations, gaining situational awareness of the environment (Step 4) is a challenge. In terms of PQS process modeling, this requires detecting processes and estimating their states from the received sensor observations.

Knowledge of the processes and their states within an environment provides the desired situational awareness (Step 5). A key complicating factor in these application domains is that many active processes are possible, and observations of the processes are interwoven, ambiguous, and unlabeled. Some observations can be missed while others can enter the data stream as noise.

Table 1 summarizes how the PQS framework applies to various application domains.

### PROCESS DETECTION

Figure 2 demonstrates the process detection concept. Figure 2a shows a simple process in which solid circles denote states  $U_1$  and  $U_2$ . The arrows between states indicate the possible transitions. In this case, the transitions from  $U_1$  to  $U_2$  and from  $U_2$  to itself are the only possi-



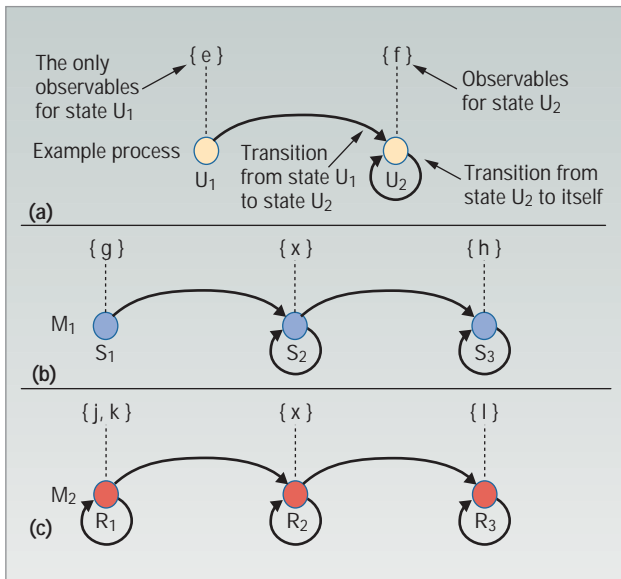
**Figure 1. PQS framework.** (Step 1) A sensed environment consists of processes with distinct states, dynamics, and observables. (Step 2) The states and dynamics of processes in an environment cannot be observed directly, but they do produce observable events and artifacts. (Step 3) The sensing infrastructure detects events and communicates them back to an analysis center. (Step 4) Sensor observations about the hidden, internal processes' states often consist of inconclusive and noisy evidence, which makes gaining situational awareness of the environment a challenge. (Step 5) Knowledge of the processes and their states within an environment provides the desired situational awareness.

bilities. State  $U_1$  is associated with the observable event  $e$ , and state  $U_2$  is associated with event  $f$ .

Researchers could use this process to model the operational status of a computer or network component such as a network interface, in which event  $e$  is a startup log entry and event  $f$  is an error message. The hidden, internal states  $U_1$  and  $U_2$  are *normal* and *failed*. The startup event is evidence that the device initially operated in a normal mode, while error messages, of which there can be many, indicate abnormal operation. An event sequence in the form  $efff$  is unambiguously associated with the state sequence  $U_1U_2U_2U_2$ , meaning that the device has entered into an abnormal or failed operating mode.

**Table 1. Process detection in various application areas.**

Application environment	Processes	States	Observables
Computer attacks	Host and network behaviors	Normal, scanned, infected, failed, trusted, hostile	Tripwire, applications logs, Snort alerts, host-based logs, file access, user access
Autonomic server farms	Server applications	Normal, degraded, failed, recovered	Performance measures, response times, Snort readings, IDS alerts
National border and physical perimeter defense	Moving objects (people, animals, vehicles)	Position and velocity	Video, infrared images, acoustic data, seismic data, electromagnetic data
Geographic region	Airborne agent diffusion and drift	Releases at times $T$ , locations $L$	Sensor detection of an airborne agent
Identity theft and management	Consumer, bank, ID thief's activities	Normal, phished, exploited	Credit reports, Web postings, database breaches, pretexts
Social networks	Business and social activity	Stages of a business or social process	Communications and transactions



**Figure 2. Different formalisms representing processes in the PQS framework. (a) This model illustrates the graphical notation used to represent a nondeterministic automaton process model, or a weak model. (b) and (c) Both of these models involve common observable events—the reason that detecting and disambiguating multiple processes become a significant challenge, even in a simple example.**

A simple rule can easily achieve situational awareness in this example—if a single error message is received, the device is in an abnormal operating mode. However, matters become more challenging when multiple process models associated with ambiguous evidence are involved.

For the process models shown in Figures 2b and 2c, the complexity of situational awareness increases even though the models themselves are relatively simple. These models, which occur naturally in computer security applications, each show three states with the possible state transitions and state-to-observation associations. A key ingredient of the two models is the ambiguity inherent in

event  $x$ , which can be associated with either state  $S_2$  of model  $M_1$  or state  $R_2$  of model  $M_2$ . All other events are unambiguously related to unique states.

If the event sequence  $gkxx$  is observed, a unique sequence of model states cannot account for these events.  $M_1$  and  $M_2$  can occupy (interleaved) states  $S_1R_1R_2S_2, S_1R_1R_2R_2$ , or other possibilities. Each consistent assignment of a subsequence of events to a process model's states is called a *track*; each consistent set of tracks that explain the entire observed event sequence is called a *hypothesis*. A set of hypotheses that can explain observed events conveys situational awareness in this example.

Future observed events can result in either an increase or a reduction in the number of hypotheses. For example, if the next event observed is  $j$ , the full observation sequence becomes  $gkxxj$ , and the only viable hypothesis is  $S_1R_1S_2S_2R_1$  because Model  $M_2$  cannot transition from state  $R_2$  to  $R_1$ —all observed  $x$  events must be associated with process model  $M_1$ . On the other hand, if the next observed event is another  $x$ , the number of hypotheses will clearly increase.

Table 2 shows the complete set of hypotheses corresponding to models  $M_1$  and  $M_2$ .

### PQS COMPONENTS

PQS is software that uses various algorithms to generate and manage hypotheses about an observed event sequence, given a set of process models. It consists of the following components.

**Track and hypothesis extension.** Given a new event, this component updates the tracks in the current set of hypotheses. Specifically, a track can be extended if its current hypothesized state can transition to a next state that can generate the currently observed event. The new tracks can be instantiated to accommodate the newly observed event, which creates a new set of hypotheses.

**Hypothesis scoring.** Because the number of hypotheses can increase exponentially, this component ranks

them according to a score that measures the merit of tracks within the hypothesis or some other metric. For example, Occam-type rules can assign high scores to the simplest hypotheses, or, if the models are hidden Markov models (HMMs), a Viterbi-type decoder can score hypotheses by the likelihood of their tracks.

**Hypothesis management.** This component keeps hypotheses with the best scores and scores above a select threshold. Possible approaches for implementing this component include applying heuristics that simply keep the highest scoring hypotheses; clustering hypotheses and keeping exemplars from each cluster of hypotheses; and maintaining a probability distribution over all possible hypotheses by updating and sampling the distribution using Markov chain Monte Carlo (MCMC) techniques.<sup>3,4</sup>

**Situation evaluation.** This component uses risk-assessment or variance-estimation computations to address ambiguous situations in which multiple hypotheses can be consistent with the observed events at any given time. For example, given multiple possible explanations for a sequence of events, it might assign reporting priority to those hypotheses that present a higher risk within the environment.

The power of the PQS approach derives from the separation of the models from the detection logic. Constructing a decision tree or a rule set to derive the possible hypotheses in Table 2, for example, is not difficult. However, how the decision tree or rule set would have to be changed if other models were added to the environment must be considered. In the PQS framework, if we add new model descriptions, the algorithmic steps depicted in Table 2 would operate as usual; in contrast, the corresponding rule sets and decision trees would have to be extensively and carefully revised to maintain the ability to disambiguate between the different possible observation sequences and resulting hypotheses. The strength of PQS is that it automates the logic of the detection and disambiguation computation, while decision trees and expert system-type rule sets must encode both the models and the detection logic simultaneously.

Even though PQS is a generic and universal approach to process detection for situational awareness issues, using the framework requires addressing several technical challenges.

- **Model derivation and description.** To date, researchers have used domain knowledge and common sense to build process models for different PQS application problems. Representations of the models have included nondeterministic finite automata (see Figure 2), HMMs, and classical state-based systems as in Kalman-filtering applications for kinematic-modeling problems.
- **Model-event scoring.** Given a subset of events and a specific process model, what are effective and efficient

**Table 2. PQS hypothesis generation showing the PQS algorithm's iterative steps on the event sequence *gkxxj*.**

Step	Observations	Tracks	Hypotheses
1	<i>g</i>	$S_1$	$\{S_1\}$
2	<i>gk</i>	$S_1R_1$	$\{S_1, R_1\}$
3	<i>gkx</i>	$S_1R_1S_2$ , $S_1R_1R_2$	$\{S_1S_2, R_1\}$ , $\{S_1, R_1R_2\}$
4	<i>gkxx</i>	$S_1R_1S_2S_2$ , $S_1R_1S_2R_2$ , $S_1R_1R_2S_2$ , $S_1R_1R_2R_2$	$\{S_1S_2S_2, R_1\}$ , $\{S_1S_2, R_1R_2\}$ , $\{S_1, R_1R_2S_2\}$ , $\{S_1, R_1R_2R_2\}$
5	<i>gkxxj</i>	$S_1R_1S_2S_2R_1$	$\{S_1S_2S_2, R_1R_1\}$

algorithms for producing a metric that captures the extent to which that process could have produced that event sequence? To date, PQS has used 0-1 scoring (possible or impossible) as in the case of finite automata, likelihoods for HMMs, and approximate likelihoods for kinematic state-space-based models in lieu of precise Kalman-filtering methods.

- **Hypothesis management.** To date, PQS applications have used simple techniques such as maintaining a small number of high-scoring hypotheses, while other researchers have used MCMC methods for kinematic tracking using sensor networks.<sup>4</sup>
- **Solution evaluation.** How can we evaluate the robustness of a solution to the deterministically synchronized sequential processes? That is, what would be the analog of a variance estimate as in traditional statistical inference? Researchers have proposed the entropy of the set of possible hypotheses as a measure of confidence in a solution, but more work remains to be done in this area.<sup>5</sup>

While our PQS implementation has demonstrated the feasibility and breadth of the approach's applicability, we will continue to address many analytic and implementation issues to improve its performance and generality.

## NETWORK SECURITY

Because enterprise-class networks require monitoring numerous sensors to manage threats, human administrators have great difficulty maintaining comprehensive real-time situational awareness around the clock. However, connecting all the sensors to PQS was simple, and using a variety of process models to correlate the event stream turned out to be effective and robust.

The test environment in this domain consisted of more than 1,000 hosts divided into several subnets, with multiple connections to the Internet. In that setting, we used IDS sensors, logs from dozens of services (Apache, Internet Information Services, syslogs, Tripwire, Samhain,

and SaMBa), and network flows.<sup>5,6</sup> Events were correlated using a large number of fairly simple process models related to both security and infrastructure failures. The models included behaviors of self-propagating worms; low and slow scans; remote administration tool deployments; malicious insider document accesses; unwanted information leakage through covert channels; and multi-stage, multicomputer intrusions. Figure 3 shows examples of some process models used in this domain.

PQS associates evidence from the sensor data stream with the possible dynamics the process models express and presents the hypothesized security and activity incidents. To prioritize tracks within a hypothesis, PQS includes a severity score. This PQS application has significantly reduced the amount of information that network administrators must review, making them more effective and efficient. In government-conducted tests, this system showed a data reduction rate of more than 200:1.

A key feature of PQS that differs from previous approaches is that it can simultaneously model, monitor, and consistently detect multiple attack models and other behaviors. By maintaining multiple hypotheses concurrently, an unlikely current explanation could become the best explanation after the system senses and processes supporting evidence.

False-positive problems are mitigated by hypotheses which, when reported to an analyst, must be internally consistent—the same evidence cannot be used to support two tracks within a single hypothesis at the same time. PQS handles the false-negative problem—missing

the detection of a true attack—by letting users build both novel sensors and models quickly and effectively. Models can be either generic or specific to suit users' requirements; since each model is a stand-alone component, integrating new models simply consists of submitting them to the PQS. In contrast, rule-based and decision tree methods require explicitly updating consistency rules and decision logic across the whole space of attacks when a new threat is added, which creates severe scaling problems.

Another area closely related to network security is the monitoring and automated repair of computer systems and their application programs.<sup>7,8</sup> As computer networks grow larger and software becomes increasingly more complex, monitoring and maintaining individual computers, especially in large-scale server farms, presents a difficult challenge. The goal is to automatically detect unusual behaviors and fix the problem before it escalates.

In this domain, using PQS to easily attach new sensor sources offers a major benefit. The PQS engine can monitor the network data to collect and process an enormous amount of information. Examples of sensor data include memory and CPU (and other general resources) usage, process-forking behavior, application logs, and network sensors (such as firewall logs and intrusion-detection alarms).

Using a specific collection of process models for this application, the PQS implementation maintains situational awareness of all hosts and servers in a network. The predictive abilities of the process models are used to estimate when deviant behavior will become a problem and recommend an appropriate action.

For example, an FTP service daemon spawning a shell (/bin/sh) process might be a sign that something is wrong, especially if it can be correlated with an IDS alarm. The obvious action would be to immediately kill the shell process and possibly also to take the FTP daemon down, since it appears to be vulnerable to a remote exploit attack.

Similarly, if the system is running a program that services network requests and leaks memory with every request it services, the host will eventually run out of memory, affecting other applications. The PQS process models predict when this will happen and preventively restart the buggy service before resource consumption becomes a problem. Figure 4 shows an example of the impact of this type of PQS-based monitoring.

### INFRARED/VIDEO TRACKING

PQS can use simple kinematic models to track the motion of multiple objects in physical space. To demonstrate this, we used PQS with a video

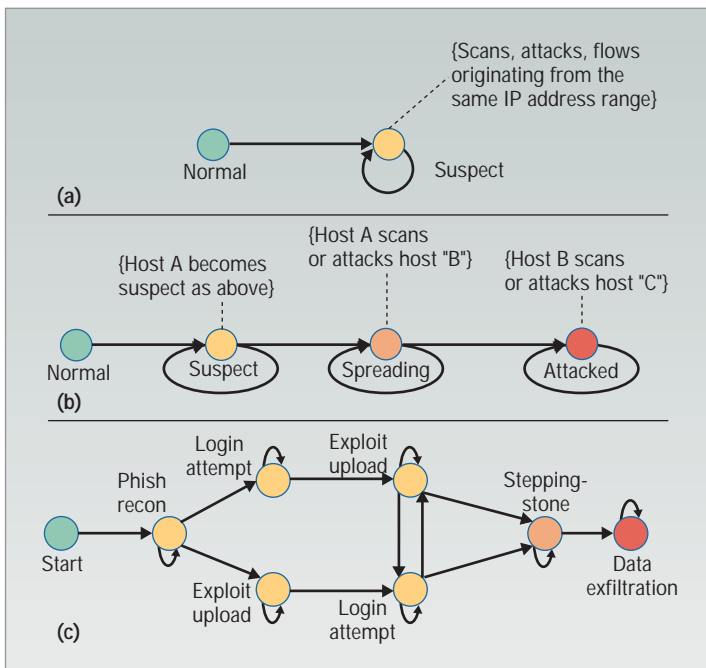


Figure 3. Process models. (a) Simple model for correlating activity at a host. (b) Stepping-stone model. (c) Multistage phishing attack model.

camera to track the movements of several fish simultaneously in an aquarium and with an infrared camera to track people inside a building.

Figure 5 shows the fish-tracking application, in which bubbles from the aerator, food, dirt particles, and an air-operated toy skeleton constituted sources of noise. The fish can be occluded by rocks and each other, resulting in ambiguity and missed event detections. The PQS implementation distinguished between the fish and other objects by correlating events with the modeled dynamics of the fish motions. Complicated video analysis and image recognition were not required.

The sensor-event stream was based on simple video-frame analysis, which first involved a frame's color segmentation. The centroids of the red regions of each frame were computed in real time (the fish were red), producing only a stream of  $(x, y)$  coordinates without other attributes. The kinematic model of a swimming fish is very simple; the "state" of a fish is  $(x_t, y_t, x'_t, y'_t)$  where  $x_t$  and  $y_t$  are positions within the field of view, and  $x'_t$  and  $y'_t$  are the velocities. The kinematic model of a swimming fish merely constrained the way that  $(x_t, y_t, x'_t, y'_t)$  could change over time. That is, the model required  $(x_t, y_t, x'_t, y'_t)$ , the state at time  $t$ , and  $(x_{t+\delta t}, y_{t+\delta t}, x'_{t+\delta t}, y'_{t+\delta t})$ , the state at time  $t + \delta t$ , to satisfy

$$|x_{t+\delta t} - x_t| \leq c\delta t x'_t, |y_{t+\delta t} - y_t| \leq c\delta t y'_t$$

and

$$x'_{t+\delta t} = \min\left(\frac{x'_{t+\delta t} - x_t}{\delta t}, \Delta_{\max}\right)$$

$$y'_{t+\delta t} = \min\left(\frac{y'_{t+\delta t} - y_t}{\delta t}, \Delta_{\max}\right)$$

The first two inequalities constrain how quickly a fish can swim between frames relative to the hypothesized fish velocity based on previous frames. The second two equalities update the hypothesized velocity with a cap on the absolute velocities in each direction.

These simple kinematic constraints summarize the model's essence. Given this model, every new  $(w, z)$  observation is matched with an existing track using the above criteria, with the "score" of the match inversely proportional to how close  $(w, z)$  is to  $(x_t + \delta t * x'_t, y_t + \delta t * y'_t)$ . Good matches are retained and become tracks. A hypothesis is therefore completely summarized by a collection of  $(x_t, y_t, x'_t, y'_t)$  state coordinates—a vector for each possible tracked object.

Since the fish could be swimming past each other with respect to the camera's field of view, multiple  $(x, y)$  centroid coordinates could match multiple existing tracks from multiple existing hypotheses. A metric or score is therefore needed to quantify how well a new measurement fits an existing model track. We experimented with Euclidean and logarithmic distances with little empirical performance variation in the results, suggesting that the models and correlation framework are relatively robust.

To take this kinematic model detection concept a step further, one of our colleagues, Alex Jordan, used a

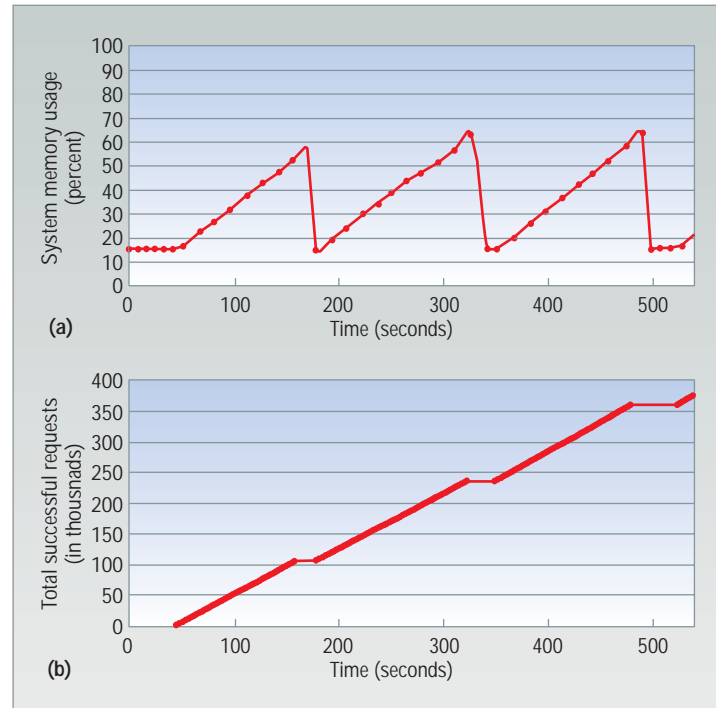


Figure 4. PQS autonomic computing application. (a) Memory consumption of a server system running a buggy daemon that is leaking memory with each serviced request. (b) Cumulative number of requests serviced by the server host. The application restarted the daemon three times during this 10-minute experiment.

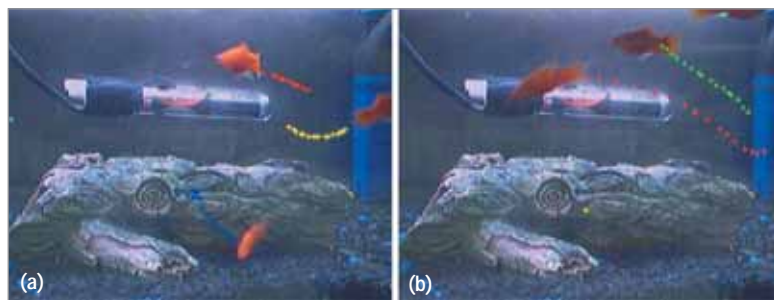
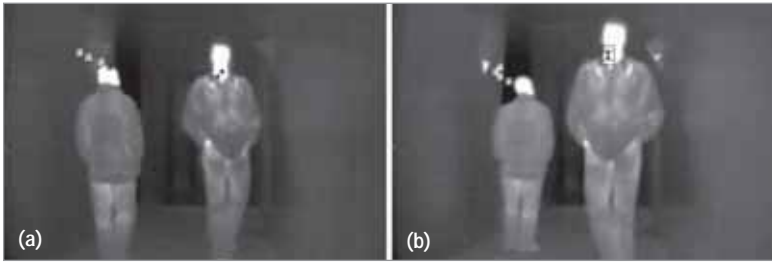


Figure 5. Fish-tracking PQS application. The application investigates the detection of schooling, pursuit, and feeding. Sample frames show fish (a) swimming past each other and (b) approaching each other. Tracks are marked in different colors.



**Figure 6. Infrared tracking application of PQS.** The process model measures changes in the total heat fingerprint and changes in their position in the frame relative to other objects and the horizon to detect whether the person is (a) approaching or (b) walking away from the camera.

second PQS engine that takes its input from the first PQS engine.<sup>9</sup> This second-level PQS engine incorporates several process models that use first-level fish tracks to detect more complex collective behaviors. This way, Jordan reliably identified when the fish were eating and when they were chasing each other.

The process model for chasing behavior, for example, has two top-level states: approach and depart. When one fish quickly approaches another, the chasing model combines two tracked fish from the first PQS into an approach state. If the second fish subsequently flees by increasing the distance between the two fish (depart), a chase detection is made. This behavior can repeat multiple times.

The entire setup was completed with a 35-gallon tank, some red platys, a 5-year-old desktop PC, and a \$100 USB webcam. Jordan wrote and debugged the PQS models within a week.

In a related application, PQS was used to track persons and objects using a thermal infrared camera. The objective was to detect whether a person was

approaching or walking away from the camera. As Figure 6 shows, the process model identifies approaching or departing objects by measuring the changes in size of their total heat fingerprint and the changes of their position in the frame relative to other objects and the horizon.

The main implementation challenge in this application was feeding the thermal camera's output to the front-end frame processing on the PQS engine computer. The video processing consisted of a simple thresholded union-find algorithm producing sets of (X, Y) coordinates as in the fish-tracking application.

This process model consisted of several states: departing, stopping, neutral move, and approaching.<sup>10</sup> Although these sample applications are relatively simple, they demonstrate the power of using PQS to create a functioning and effective situational-awareness system in a domain quite different from cybersecurity.

### DYNAMIC SOCIAL NETWORK ANALYSIS

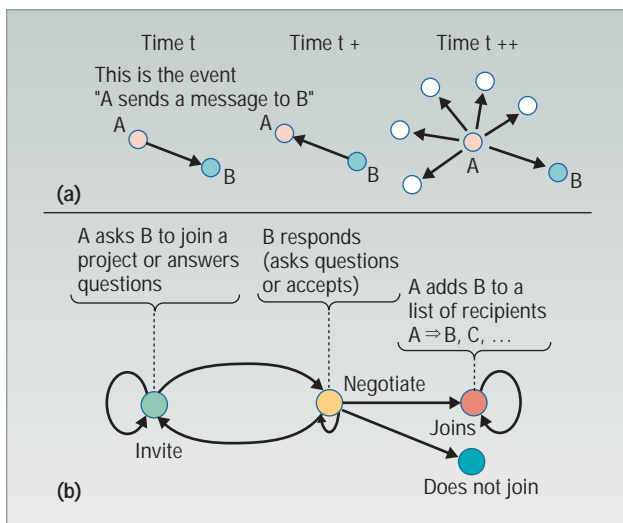
Detecting social and business processes in networks of people and organizations serves as another example of effectively applying PQS to a complex environment. In this domain, researchers have observed communications events and other transactions between people such as e-mail, phone calls, postal mail, meetings, instant messaging, and financial transactions to detect the processes being undertaken and the roles people play in the network.

For example, can the timing and propagation of a CEO's communications identify the person that holds that position? Is it possible to determine a person's business process (product planning, purchasing, or personnel recruiting) based on the detected temporal and structural event correlations?

Within the PQS framework, the assumption is that people engage in various social or business processes with purpose. That is, they are trying to accomplish something by communicating and participating in financial dealings or meetings.

For example, building a team is a common social and business process in which an initiator contacts one or more people with a proposal to join a group. The team could be formed to write a joint proposal, develop a new product, or have a picnic. The initiator, A, contacts a person, B, with the proposal, and B either agrees or asks for more details. As Figure 7 depicts, this can lead to a sequence of exchanges.

A major difference between the PQS approach and traditional social network analysis is that PQS models the dynamic processes that exist within a social network, not merely the static structural artifacts—such as who knows whom—of such a network. PQS can use the



**Figure 7. Social network analysis.** The PQS framework can model the relationships between events and business or social processes. (a) Temporal sequence of communications. (b) Business process model for recruiting a team.

temporal nature of communications and transactions to extract processes.

A second-tier PQS model identifies the role that a specific person, or actor, plays in a network or process.<sup>11</sup> Communications events are associated with the states in the first-tier process models that indicate the roles an actor plays. This can change over time, or an actor may play multiple roles at once. Wayne Chung and coauthors provided additional details of the PQS approach to dynamic process-based social network analysis.<sup>11</sup>

## CHEMICAL PLUME TRACKING

Airborne agent plume tracking is an area of increasing interest and concern. An airborne agent release, whether it is smoke from a fire, pollution from an industrial plant, or a toxic agent released with malicious intent, will spread according to the physical dynamics of a gas in a windy environment. Drift-diffusion partial differential equations can describe these dynamics.

The sensing infrastructure in this application domain is a distributed network of sensors that can detect an agent's presence. As Figure 8 shows, the sensors report detections, which can be very noisy and highly granular. Our simulations have used noisy binary sensor models; that is, a sensor can detect the agent without concentration-level readings, and the detection itself is subject to noise. Time, sensor location, and the sensor's detection status are the reported events of this simulation.

Sensor reports are collected continuously at a central site running a PQS engine with drift-diffusion models of airborne agents. Our models require continuous monitoring of the wind velocity to determine the drift component of the time-varying model.

PQS can quickly and directly support developing multiple hypotheses about how the agent is spreading, the location of estimated release points, and where the plumes are heading. The multiple processes correspond to the agent's possible release points.<sup>12</sup> Correct estimation of the release points and future evolution of the plumes allows for both investigation and response.

The introduction of affordable large-capacity disk drives and modern database management systems based on Structured Query Language (SQL) revolutionized data processing in the 1970s. A major result

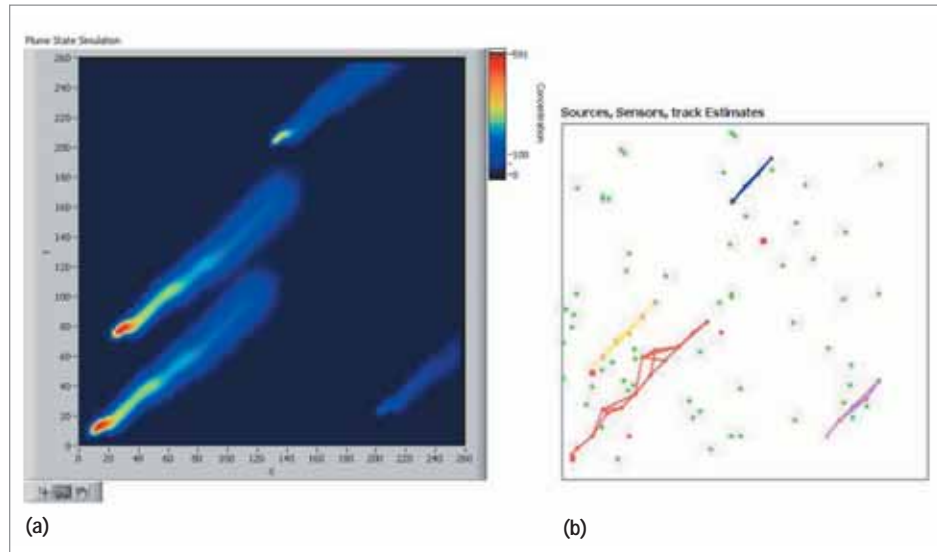


Figure 8. Airborne agent plume tracking. (a) Two releases of an airborne agent subject to wind drift and natural diffusion. (b) PQS implementation showing a field of sensors with their detection status and estimated and reconstructed plume tracks.<sup>12</sup>

of that revolution was that database-applications developers could focus on their application logic and programming and disregard the routine bookkeeping and overhead of file and record access in sequential files.

We believe that the PQS concepts and software implementations can have the same revolutionary effect on sensor network data processing and the resultant situational-awareness applications built on top of sensor networks.<sup>13</sup> Application developers can use PQS to build situational awareness systems faster, better, and with fewer errors.

Another key innovation of the PQS approach is that it is patently process-based, encouraging developers to think about the dynamics, not just the static artifacts, of an environment. By explicitly modeling the dynamics of multiple processes in an environment, the ambiguity of event-to-process model associations introduces computationally difficult—that is, NP-complete—matching problems that researchers cannot generally solve efficiently. Much work remains to scale PQS processing so that it remains effective within acceptable efficiency constraints when dealing with larger problems.<sup>14</sup>

Additional references and a preliminary release of our research PQS software can be found at [www.pqsnet.net](http://www.pqsnet.net). ■

## Acknowledgments

This work was a project of the Institute for Security Technology Studies at Dartmouth College and was supported in part by ARDA under Grant F30602-03-C-0248, DARPA Projects F30602-00-2-0585 and F30602-98-2-0107, and Award number 2000-DT-CX-K001 from the US Department of Homeland Security, Science and

Technology Directorate. All opinions expressed are those of the authors, not the sponsoring agencies.

## References

1. M. Stonebraker, U. Cetintemel, and S. Zdonik, "The 8 Requirements of Real-Time Stream Processing," *ACM SIGMOD Record*, Dec. 2005, pp. 42-47.
2. Y. Yao and J. Gehrke, "The Cougar Approach to In-Network Query Processing in Sensor Networks," *ACM SIGMOD Record*, Sept. 2002, pp. 9-18.
3. S.S. Blackman, *Multiple-Target Tracking with Radar Applications*, Artech House, 1986.
4. S. Oh et al., *A Scalable Real-Time Multiple-Target Tracking Algorithm for Sensor Networks*, tech. report UCB/ERL M05/9, Univ. of California, Berkeley, 2005.
5. V.H. Berk and N. Fox, "Process Query Systems for Network Security Monitoring," *Proc. SPIE—Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense IV*, 2005, pp. 520-530; <http://people.ists.dartmouth.edu/~vberk/papers/berkfox5778-75.pdf>.
6. O. Kreidl and T. Frazier, "Feedback Control Applied to Survivability: A Host-Based Autonomic Defense System," *IEEE Trans. Reliability*, Mar. 2004, pp. 148-166.
7. C. Roblee, V. Berk, and G. Cybenko, "Implementing Large-Scale Autonomic Server Monitoring Using Process Query Systems," *Proc. 2nd Int'l Conf. Autonomic Computing*, IEEE CS Press, 2005, pp. 123-133.
8. G. Jiang, H. Chen, and K. Yoshihira, "Discovering Likely Invariants of Distributed Transaction Systems for Autonomic System Management," *Proc. IEEE Int'l Conf. Autonomic Computing*, IEEE CS Press, 2006, pp. 199-208.
9. A.B. Jordan, "Models for Tracking and Level 2 Fusion," master's thesis, Thayer School of Eng., Dartmouth College, 2005.
10. A. Barsamian, V.H. Berk, and G. Cybenko, "Target Tracking and Localization Using Infrared Video Imagery," *Proc. SPIE Unattended Ground, Sea, and Air Sensor Technologies and Applications VIII*, vol. 6231, May 2006, pp. 1-7; [www.ists.dartmouth.edu/library/246.pdf](http://www.ists.dartmouth.edu/library/246.pdf).
11. W.W. Chung et al., "Identifying and Tracking Dynamic Processes in Social Networks," *Proc. SPIE Sensors, and Command, Control, Communications, and Intelligence (C3I) Technologies for Homeland Security and Homeland Defense V*, vol. 6201, May 2006, pp. 1-12.
12. G. Nofsinger and G. Cybenko, "Distributed Chemical Plume Process Detection," *Proc. Military Comm. Conf.*, IEEE Press, 2005, pp. 1076-1082.
13. V.H. Berk, "Process Query Systems," doctoral dissertation, Leiden Univ., 2006.
14. V. Crespi, G. Cybenko, and G. Jiang, "The Theory of Trackability with Applications to Sensor Networks," submitted to *ACM Trans. Sensor Networks*, Aug. 2006; also available as tech. report TR2005-555, Computer Science Dept., Dartmouth College, 2006.
15. D. Hernando, V. Crespi, and G. Cybenko, "Efficient Computation of the Hidden Markov Model Entropy for a Given Observation Sequence," *IEEE Trans. Information Theory*, July 2005, pp. 2681-2685.

IEEE  
Computer  
Society  
members

save  
25%

on all  
conferences  
sponsored  
by the  
IEEE  
Computer  
Society

[www.computer.org/join](http://www.computer.org/join)

*George Cybenko is the Dorothy and Walter Gramm Professor of Engineering at Dartmouth College. His research interests include distributed information, control systems, computer security, and signal processing. Cybenko received a PhD in mathematics from Princeton University. He is a Fellow of the IEEE, a member of SIAM, and serves on the boards of the IEEE Computer Society and the Computing Research Association. Contact him at [gvc@dartmouth.edu](mailto:gvc@dartmouth.edu).*

*Vincent H. Berk is a lecturer and research scientist at Dartmouth College and a cofounder of ProQueSys LLC. His research interests include computer network security, high-performance computing, and complex systems software design. Berk received a PhD in computer science from Leiden University. He is a member of the IEEE, the ACM, and the MAA. Contact him at [vhb@dartmouth.edu](mailto:vhb@dartmouth.edu).*