PROJECT SUMMARY: Software Infrastructure for Seamless Network Simulation and Deployment

There is a growing rift in the practice of networking research between simulation and system building. Network simulators have generally been good for research. They provide an easy and inexpensive way to test new protocols, new network elements, and changes to existing protocols; allow for controlled experiments and independent confirmation; and represent a shared knowledge base networking researchers can draw from and add to. These advantages have led, over time, to a significant increase in the fraction of networking systems work that has been evaluated predominantly, or even exclusively, in simulation.

The end goal of networking systems research is still widespread deployment, however, and simulation, while powerful, cannot substitute for deployment experience. The Internet's wild heterogeneity is essentially impossible to simulate [23]. Important emergent properties of protocols and systems become clear only in the unexpected conditions of deployment. Carefully chosen simulation parameters can even produce results not seen in the real world. The increasing dependence of research on simulation is perfectly understandable; why implement your system twice, once for simulation and once for deployment, if you don't have to? Unfortunately, while tempting, overreliance on simulation is both bad for research and bad for the Internet. Without grounding in empirically observable Internet behaviors, research can get sidetracked into problems that are relevant only in simulation, ignoring problems that are actually hampering the Internet's evolution.

This proposal will bridge the gap between simulation and deployment by introducing a common software infrastructure for simulation and deployment of network systems. This infrastructure will allow use of *the same code* both in simulation and in deployment. We want to make it so easy to write code usable in both situations that there is no longer any excuse for testing systems only in simulation. The products of our research will be a set of reliable, well-maintained tools useful for students and the research community. Much like Emulab, the Utah network testbed [10], our aim is the betterment of the practice of network research.

The basis of our infrastructure will be the Click modular router [16] and the *ns* network simulator [21]. *Ns* is the predominant network simulator used in networking research today. Click router configurations are comprised of configurable, interconnected *elements*, each of which performs a simple packet processing function. Click's configurability and code reuse are ideal matches for the flexibility required to rapidly build and modify a simulated network system. Click is already suitable for deployment—a Click-based IP router can achieve more than 200 Mbit/s forwarding rates on a conventional PC. However, elements generally behave independently of their context, so, for example, devices are easily replaced by traffic generators; and a support library makes elements portable between OS kernels and user level. Click is thus also suitable for simulation. We will make it possible to use Click elements and router fragments in simulations either by extending Click alone, or, more likely, by seamlessly integrating it into *ns*.

The work of our proposal divides into two areas: building the simulation–deployment infrastructure, and evaluating its usefulness by application. To build the infrastructure, we must evaluate different software architectures (Click alone, Click embedded in *ns*, or *ns* embedded in Click), then implement one or more of them. This will require developing new components for Click and, probably, *ns*; working extensively on integration; and addressing research problems in the usability of the resulting tools. To evaluate and guide our work, we will simultaneously use our infrastructure-in-progress to address networking research problems clearly suffering from lack of deployment experience. These problems include active queue management algorithms, wireless networking, and aggregate-based congestion control. Our goal is not fundamental research in these areas, but adding deployment tools and experience to the research that already exists.

Part C
PROJECT DESCRIPTION

There is a growing rift in the practice of networking research between simulation and system building. Network simulators have generally been good for research. They provide an easy and inexpensive way to test new protocols, new network elements, and changes to existing protocols; allow for controlled experiments and independent confirmation; and represent a shared knowledge base networking researchers can draw from and add to. These advantages have led, over time, to a significant increase in the fraction of networking systems work that has been evaluated predominantly, or even exclusively, in simulation.

The end goal of networking systems research is still widespread deployment, however, and simulation, while powerful, cannot substitute for deployment experience. The Internet's wild heterogeneity is essentially impossible to simulate [23], and important emergent properties of protocols and systems sometimes become clear only in the unexpected conditions of deployment.

This proposal will bridge the gap between simulation and deployment by introducing a common software infrastructure for simulation and deployment of network systems. This infrastructure will allow use of *the same code* both in simulation and in deployment. We want to make it so easy to write code usable in both situations that there is no longer any excuse for testing systems only in simulation. The products of our research will be a set of reliable, well-maintained tools useful for students and the research community. Our aim is the betterment of the practice of network research.

The basis of our infrastructure will be the Click modular router [16] and the *ns* network simulator [21]. The work of our proposal divides into two areas described below: building the simulation–deployment infrastructure, and evaluating it by building applications that use it.

1 Problems with a simulation-only world

The *ns-2* network simulator project [3, 21] and its predecessors have been in active development and use for more than 12 years, with generous support from NSF and DARPA. *Ns* has been wildly successful; for example, it was used in many of the 2001 SIGCOMM conference papers. Powerful network animators for *ns* make simulation accessible even for undergraduates, and provide powerful visualization for everyone. *Ns*'s collection of network protocols, queue management schemes, link characteristics, and accumulated network knowledge is unparalleled. While *ns* has at least its share of flaws, it has undoubtedly improved the state of networking research.

Unfortunately, *ns*'s success has led to a divergence from deployed, and deployable, networking systems. There is no way to turn an *ns*-implemented protocol or module into a real implementation. *Ns* was designed exclusively for simulation, so primitives like packet passing behave incompatibly from the corresponding primitives in real routers, or even Linux- or BSD-based PC routers. An *ns* module can depend on floating point or high-quality random number generation, or ignore how much per-packet processing an algorithm requires. *Ns* cannot simulate systems as large and heterogeneous as the Internet [23], so an algorithm that works in simulation may fail in practice. Nevertheless, many researchers continue to write code exclusively for *ns*. Their reasons: *ns* is powerful, widely available, and relatively well-understood, and they do not necessarily have time to implement the same algorithm twice, in different settings, for what they think might be small gains.

While the power of simulations to explore functionality not present in the current Internet is one of its main strengths, an unnecessary divergence of simulation studies from studies of the empirically observable Internet can be bad both for research and for the Internet. For example, without grounding in empirically observable Internet behaviors, research can get sidetracked into problems that are relevant only in simulation. (It is also possible for studies in experimental testbeds to get bogged down in unrealistic assumptions about topology and traffic patterns, but experimental testbeds can be opened up to communicate with the outside world.) For an example, we'll consider the research history of active queue management.

1.1 Case study: active queue management

The original paper on RED queue management [9] initiated a long series of papers on active queue management schemes for routers [2, 7, 8, 11, 17–19]. Combined with explicit congestion notification [24], AQM schemes are expected to provide a significant improvement in performance over the current generation of Internet routers. Yet active queue management remains undeployed on most networks, and some researchers claim that it doesn't actually work better in practice than the usual drop-tail [13, 14]. Why this gap between the best published work, the best opinions of the research community, and practice?

Because the research was performed exclusively in simulation. Of the different proposals for AQM, we believe that only RED has actually been implemented in mass-market routers, or studied in deployment settings. However, the source code for Cisco and Juniper RED is not available, and many of the published deployment studies [5, 14, 26] relied on experimental, hand-crafted implementations. These studies, therefore, are difficult to compare with simulation results, or with each other.

To address this issue, we (Padhye and Floyd, with assistance from Kohler) began a project to adapt published AQM schemes for deployment, with the end goal of comparing schemes in a real network setting. This taught us a lot about the limitations of designing networking code exclusively for simulation.

- Several schemes use very precise floating-point arithmetic, complicating implementation in an OS kernel.
- The simulation code for the different schemes was difficult to port even to a single simulator, since the schemes originally used different versions of *ns*.
- Published simulation results proved incomparable because of differences in parameter settings.
- We have now simulated versions of several AQM schemes, and explored their relative performance in the same set of simulation scenarios. The trade-offs between AQM schemes and drop-tail queueing—utilization vs. drop rates vs. delay, and so forth—are proving surprisingly hard to characterize. Benefits claimed by some AQM schemes can be achieved only in limited contexts. The skepticism network implementors have shown for simulations turns out to be justified: carefully chosen simulation parameters can produce results not seen in practice, and nothing subsitutes for deployment testing.

Our experience with AQM shows clearly that networking schemes should not be tested exclusively in simulation. But we shouldn't throw the baby out with the bathwater: simulation is a vital tool in the network researcher's toolbox. We need a way to seamlessly move algorithms and code between simulation and deployment.

2 From deployment to simulation

Moving code from a simulator into a deployment environment presents major challenges. As we have seen, simulation code assumes a rich library that isn't available in, say, an OS kernel. More promising is the other direction: moving deployment code into a simulator. It's relatively simple to port code to a less restricted, user-level simulation environment.

Element name	Function
DecIPTTL	Decrement IP packet's time-to-live.
UDPIPEncap	Encapsulate packets in UDP and IP headers.
StaticIPLookup	Look up IP routes from a static table.
ToDevice	Send packets to a network device.
Queue	Store packets in a FIFO queue.
FromDump	Read packets from a *tcpdump* file.
Shaper	Shape packet flow to a given maximum rate.

Figure 1: Some Click element descriptions.

One networking system that has taken that path is *x*-sim [4], the *x*-kernel simulator. The *x*-kernel is an object-based framework for implementing network protocols [12, 27]. C-language modules correspond to protocols, and a compile-time graph plugs modules together—saying, for example, that IP should send packets via ARP over Ethernet. A simulation in *x*-sim contains several protocol stacks, one per simulated machine. The lowest protocol in each stack connects to a special *sim* protocol, which manages network channels.

With *x*-sim and the *x*-kernel, one piece of network protocol code serves both for simulation and for real implementation in a specialized kernel, exactly like we want. But design choices oriented towards network protocols limit this particular system's applicability. *x*-kernel configurations must be acyclic and layered, right for protocol stacks but not necessarily for routers. Coarse-grained *x*-kernel components aren't well-suited for reuse in new contexts. Mandatory features like session nodes—there's one session node per end-to-end flow—are superfluous to many networking components, such as routers or stateless firewalls.

Dummynet [25], while superficially not a simulator, demonstrates how much people value the ability to move between simulation and deployment. Dummynet adds virtual links with configurable bandwidth, delay, and loss characteristics to FreeBSD; users can connect virtual interfaces into arbitrary topologies with Dummynet links. This makes it possible, although difficult, to test kernel protocol implementations, and user applications, on modest simulated networks. The Dummynet model, however, does not scale well to large topologies or complex, time-dependent simulations.

3 The Click modular router

We propose to develop a simulation framework for Click [16], a modular networking system originally designed for extensible, fast PC routers. Click is especially suitable for simulation because of its highly developed, fine-grained component system, a large existing library of components, its application-neutrality, and the existence of tools for automatically analyzing and transforming configurations.

A Click router consists of fine-grained components called *elements*. A programmer can just add, remove, or rearrange a couple elements to adjust a router's functionality. Elements' simple, focused specifications make them easy to understand and reuse. The Click distribution comes with more than 250 elements (see Figure 1 for a couple of them). If no existing element is appropriate, an extensive support library makes it easy to write new elements: most elements distributed with Click take less than 250 lines of C++. Although most existing elements follow a particular, fine-grained style, users can easily write elements in any style: the system is application-neutral. Click routers run inside Linux or FreeBSD kernels, or at user level. In a Linux kernel, using polling device drivers, a Click router can sustain a forwarding rate of 400,000 minimum-size Ethernet packets per second, or about 200 Mbit/s, on a 700-MHz Pentium III PC. Most elements compile and run equally well anywhere; the support library hides differences between operating systems, and between

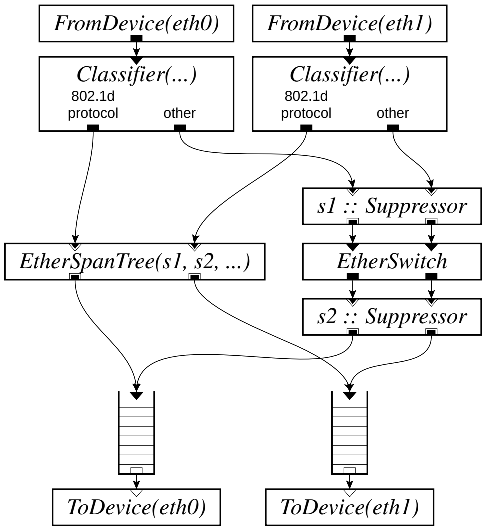


Figure 2: A Click Ethernet switch configuration.

```
fd0 :: FromDevice(eth0);    fd1 :: FromDevice(eth1);
c0, c1 :: Classifier(...);  q0, q1 :: Queue;
td0 :: ToDevice(eth0);      td1 :: ToDevice(eth1);

est :: EtherSpanTree(s1, s2, ...);
s1, s2 :: Suppressor;       sw :: EtherSwitch;

fd0 -> c0[0] -> [0]est[0] -> q0 -> td0;
fd1 -> c1[0] -> [1]est[1] -> q1 -> td1;
c0[1] -> [0]s1[0] -> [0]sw[0] -> [0]s2[0] -> q0;
c1[1] -> [1]s1[1] -> [1]sw[1] -> [1]s2[1] -> q1;
```

Figure 3: A Click-language definition for Figure 2.

the kernel and user level.

Figure 2 shows a simple Click router configuration, an IEEE 802.1d-compliant learning, forwarding Ethernet bridge. Since the components are fine-grained and have sensible names, much of the router's functionality can be read off the diagram. Of the Ethernet-switch-specific elements, *EtherSwitch* provides a simple learning, forwarding core—it would be usable on its own as a hub—while *EtherSpanTree* implements the 802.1d spanning tree protocol. Users write Click router configurations with a declarative language that says how elements should be connected together; Figure 3 shows one such definition for the switch configuration.

4 Click and simulation

Now, say that we have a network application that we would like to simulate. Should we use *ns*, or some other dedicated simulation platform; Click, or some other deployment platform, in simulation mode; or some combination of the two? We have argued that the *ns*-only approach has disadvantages if deployment is ever desired. Setting it aside, is there a clear choice between the other approaches?

Element name	**Function**
RatedSource	Generates traffic at a specified rate.
TimedSource	Generates traffic off a timer.
TimedSink	Dequeues packets off a timer.
Counter	Counts packets.
SetCycleCount	Annotates packets with cycle counts.
CycleCountAccum	Keeps track of cycle count deltas.
DriverManager	Turns traffic sources off and on, and otherwise manages the router configuration, as specified.
Print, *IPPrint*	Print packets.
CheckIPHeader	Checks an IP packet's validity.
TCPConn, *TCPAck*, ...	Manage fake TCP connections.

Figure 4: Some Click elements useful for simulation and debugging.

No; in fact, the approaches are good at different things. A Click-only simulation platform is simpler than a combined platform, and possibly faster at some tasks, since there is no need for translation between different representations for concepts like packets. However, a combined platform could make use of the large installed base of *ns* extensions and domain knowledge, while still providing a trivial deployment path for some simulation code.

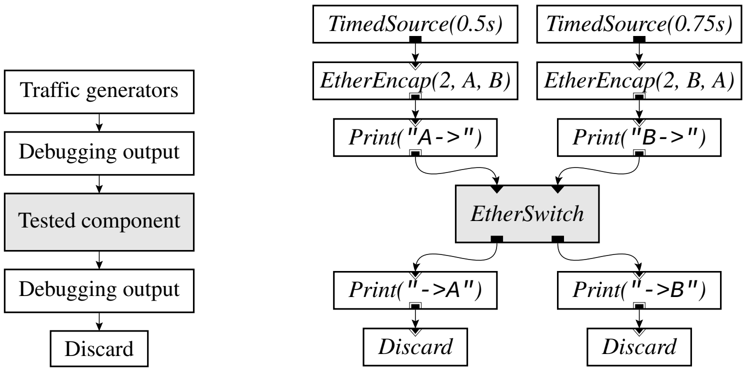
Our first task, should this proposal be funded, is to evaluate these competing approaches in detail and choose whether to implement one, or both. This section begins that work by describing some of the pros and cons.

4.1 Click as a simulator

Click itself is already well-suited for running small-scale simulations; Figure 4 lists some of its existing traffic generation, measurement, and debugging elements. This Click-only approach has several advantages over *ns*-only simulations:

- Click's fine-grained component model makes it possible to test, and measure, networking components in isolation.
- Click simulations can include real router configurations. Only the device-handling elements need differ from a deployment setting. The network elements behave exactly as they would in deployment, so simulation becomes more accurate.
- Performance testing is not very meaningful with simulator code, but easy and relevant with Click.
- Click simulation elements, such as traffic generator elements, can also run in deployment settings. This makes it easy to compare the results of a simulation with the same traffic on a real network.
- Click's declarative programming language supports analysis and source-to-source transformations, which lets us talk about static properties of network configurations (Is this loop-free? What is the network's diameter?).

There are also disadvantages. We will address some of these as part of the work of this proposal. Others may turn out to be inherent difficulties with Click-only simulation, best addressed by moving to a mixed Click-plus-*ns* model.



a. Generic plan b. Testing *EtherSwitch*

Figure 5: Simulation configurations for component testing, in general and for *EtherSwitch*.

- *Ns*'s extensive and powerful library would have to be ported, at least partially, to Click elements. This would have advantages, such as enabling the use of that library in deployment settings, but it is a large task.
- Even more importantly, the research community's extensive experience with the existing *ns* simulator should not be simply thrown away. People will continue to write *ns* simulations; it should be possible to compare those simulations with Click simulations within the same framework.
- Click's support for user abstractions would need improvement before large-scale simulations became practical. This might include programs for checking Click configurations for errors, as well as extensions of Click's input language. *Ns*'s choice of Tcl as a command language, while too low-level for some, is strictly more powerful than what Click provides.
- Finally, many kinds of simulation don't require packet-level dynamics. For those simulations, modeling invidual packets simply slows the simulation down; the user might want to abstract the simulation to a higher level [3]. Click, however, was designed explicitly for packet manipulation. Abstracted simulations are probably not appropriate for deployment, but it would be extremely useful to let the user choose the tradeoff between precision and simulation speed, without losing deployability.

The rest of this section shows what a Click simulator might look like by presenting several simulation configurations. We focus first on the Ethernet switch of Figure 2, and on things Click can already do. The last subsection describes our future plans for Click simulation.

4.2 Component testing

To test a single component, we would like to remove that component from its context, as far as is practicable, and add traffic generators, debugging output, and counters (Figure 5a). This is impractical in a monolithic simulator like *ns*; *ns* simulates networks, not components. In Click, however, it is natural. For example, Figure 5b shows how we might test *EtherSwitch* in isolation. The *Print* elements will produce output like this:

```
A->:   83 | 01010101 01010000 00000000 00025261 6e646f6d 2062756c
->B:   83 | 01010101 01010000 00000000 00025261 6e646f6d 2062756c
B->:   83 | 00000000 00000101 01010101 00025261 6e646f6d 2062756c
->A:   83 | 00000000 00000101 01010101 00025261 6e646f6d 2062756c
```

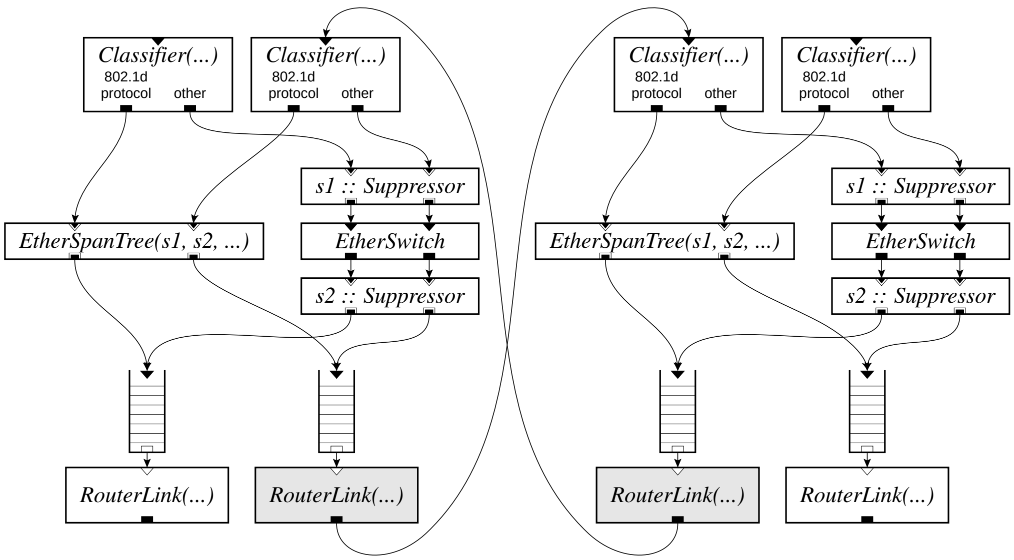


Figure 6: A portion of a simulated network of Ethernet switches. Physical links between switches are explicitly represented by *RouterLink* elements. Here, for example, the left and right switches are linked.

A user or program can easily examine this output and check that all is well.

4.3 Performance testing

Performance tests are essentially forms of simulation, involving traffic generators, measurement, and repeatable experiments possibly independent of real network conditions. Obviously, traditional network simulators don't handle this kind of simulation. Click easily supports both localized tests of a component's performance, and tests of how well a router can perform.

Microbenchmarking a Click component, or router fragment, looks a lot like testing it. Take Figure 5, replace the debugging elements (*Print*) with measurement elements (like cycle-count testers), and you have a microbenchmark setup. The transformation is so easy because of Click components' fine granularity.[1]

Of course, no one-machine simulation can tell you how a configuration will perform in practice, since, for example, there are no device interactions. To change a Click simulation into a real-world test, one can simply move the traffic-generation elements onto other machines, and have them actually send their packets over the network. The traffic is thus exactly the same as in the simulation, so the simulation and deployment settings can be used to check one another. This turns Click into a network emulator, a simulator that can communicate with a live network; Dummynet [25], NIST Net [15], and the *ns* emulator [6] are other examples. Click, however, can be significantly more flexible and powerful.

4.4 Simulating a network

Now, we move beyond component and performance testing to simulation of a network of devices. For example, how might we simulate a network consisting of several Ethernet switches connected in some topology?

[1] Some performance counters also demonstrate one of Click's secondary advantages. Intel Pentium performance counters may be used only when the processor is in supervisor mode, so some measurement configurations must run inside the kernel. But that is no problem: Click is perfectly happy to run simulations inside the kernel, just as it's happy to run real routers there.

In the simplest case, we can start with several copies of the Ethernet switch configuration; to represent the topology, we simply hook the configurations together. Figure 6 shows an example: *RouterLink* elements have replaced the device elements, linking independent configurations into a network model. Different *RouterLink*-type elements might model channels with different bandwidths and delay and loss characteristics. Traffic sources and sinks would plug into the *RouterLink*s. This Click simulation, unlike a conventional simulation, exactly models the relevant network elements' behavior (except for device interactions), making the results more precise.

4.5 Discussion

We've shown that Click-only simulations are both possible and, in some ways, more powerful than *ns*-only simulations. Four problems stand out: Click's lack of simulation components, its usability as a simulator, its effectiveness for large simulations, and the static nature of its configurations. Secondary problems include Click's handling of time and event tracing. This section discusses how we might address these problems.

Simulation components Click contains several elements useful for simulation—such as *RatedSource*, a constant-rate traffic generator—but nowhere near the panoply of choices available in *ns*. We plan to reimplement some of *ns*'s functionality in Click, as appropriate (we may abandon the Click-only simulation idea early, in favor of a mixed Click-and-*ns* model). One particularly glaring absence is a decent TCP traffic generator. This project should require more development than research, although dividing functions into reusable components often requires careful consideration.

Usability Figure 5b has one test element and ten support elements. This demonstrates one potential disadvantage of a fine-grained component system: it takes a lot of components to get anything done. Similarly, Figure 6 is dominated by switch components; the topology is difficult to read off the graph, and perhaps difficult to write in the first place. Both of these are problems with Click's *usability* as a simulator. This issue is critical: if Click is hard to use, it will fail, particularly for large-scale simulations.

We plan to apply programming language technology, and careful system design, to mitigate these issues. The main insight is to treat router descriptions as programs in a real programming language, with associated abstraction mechanisms, optimizers, and language processing tools. Compare this to *x*-sim, whose configuration language is impoverished, or *ns*, whose general-purpose scripting language is hard to analyze or manipulate.

Two examples should demonstrate the usefulness of programming language technology in router configurations. The first, *compound elements*, is an abstraction mechanism analogous to functions or macros for router configurations. Compound elements let users abstract away portions of a router configuration, clarifying how the configuration fits together at a high level. Figure 7 shows this with a compound-element definition of the component testing configuration. The second, the *click-combine* tool, automatically constructs network configurations like Figure 6, given definitions of the component configurations and a topology. *Click-combine* is basically a source-to-source transformation; it reads several router configurations in the Click language, combines them into a network by following user instructions, and outputs the resulting configuration in the Click language. Assuming `sw.click` contained the switch configuration, something like Figure 6 might be generated by this simple invocation:

```
% click-combine A=sw.click B=sw.click A.eth1=B.eth0 B.eth0=A.eth1
```

The source-to-source nature of *click-combine* deserves more comment. Compare it with conventional simulators, where the network topology is kept distinct from the network element configurations. While

```
// These element class definitions abstract the simulation boilerplate.
elementclass Source { $interval, $src, $dst |
  TimedSource($interval) -> EtherEncap(2, $src, $dst)
    -> Print("$src->") -> output;
}
elementclass Sink { $dst |
  input -> Print("->$dst") -> Discard;
}

// That done, 4 lines define the configuration.
AddressInfo(A 0:0:0:0:0:0, B 1:1:1:1:1:1);
sw :: EtherSwitch;
Source(0.5s, A, B)  -> [0]sw[0] -> Sink(A);
Source(0.75s, B, A) -> [1]sw[1] -> Sink(B);
```

Figure 7: A Click-language definition of Figure 5, using compound elements.

superficially natural, this distinction complicates some forms of analysis: for instance, a packet reachability analysis (where might packets end up if they started at X?) must consider two fundamentally different abstractions. When everything is modeled in one simple, declarative router configuration graph, analysis becomes correspondingly easier.

As part of our work on this proposal, we plan to make Click more usable in the following ways.

- Enhance the compound element abstraction. For example, it could support more flexible forms of argument passing.
- Investigate other language abstractions, such as iteration ("create 100 copies of element X").
- Build more language tools like *click-combine*, particularly tools that check configurations for programming errors and for network properties like loop freedom.
- Develop visualization tools for Click configurations, enabling easier development and maintenance of Click-language programs.
- Design and build community mechanisms for sharing Click elements, configurations, and configuration fragments. This is essentially an education function: there are already 260 elements in Click; how do you know the name of the one you want? How can we teach you what's available?

Large simulations Click was designed to run router configurations with hundreds of elements, not simulations with thousands of component "routers". While element overhead is relatively small—around 70 bytes per element, depending on the length of the element name—it is not insignificant; and there is a usability overhead associated with such large configuration files. We will begin by investigating how far Click's current infrastructure can be pushed—the maximum number of elements that keep the system responsive, for example. This may reveal bottlenecks that we need to fix. After that point, we will decide whether to push Click further with more radical changes—developing lightweight elements, perhaps, or an alternative simulation driver that sacrifices speed on small configurations for robustness on large ones.

Dynamic configurations Click router configurations are static. A configuration file describes the complete set of elements that will exist for the lifetime of the simulation. Elements can turn themselves on and off, of course, but elements cannot be created or destroyed. This has advantages—offline analysis programs can look at the complete state of a simulation, for example—but it also prevents some potentially-useful structures. Unlike the *x*-kernel, for example, Click does not support elements that are specific to a given end-to-end flow. We plan to investigate different limited forms of element creation and deletion, with one constraint: they must not prevent configurations from being statically analyzed before they are run. This is a research problem, with no immediately obvious solution; it may turn out that any solution would be uncomfortably complex, or that no solution is necessary for common simulation configurations—that elements turning themselves on and off is sufficient.

Secondary issues Click currently has a relatively inflexible idea of time, since it was designed for real-time, deployment use. In a simulation, time becomes much more flexible; we will investigate ways of combining flexible time with real time, without overly slowing down Click for deployment.

Most network simulators feature unified event tracing facilities; *ns* event traces, for example, plug right in to the *nam* network animator. Click, on the other hand, currently leaves tracing up to independent debugging elements like *Print*. We will evaluate this issue and decide whether an integrated solution is necessary. Our goal is to make Click simulations easily able to generate *nam*-compatible traces.

The Emulab project at the University of Utah [10] has used Click for some of its projects already, but a tighter integration with Emulab would provide a fantastic path for large-scale experimental deployments.

5 Combining Click and *ns*

This section turns to another model for combining simulation and deployment: making *ns* and Click cooperate. This model makes sense whether or not Click becomes a reasonable simulation platform for small-to-medium simulations. It, at least potentially, allows each system to do what it's best at: *ns* handles inter-node topology, complex traffic generation, and parts of the network where precision is not so important, while Click handles the core functionality being studied with an eye towards future deployment. It also lets current *ns* users gently upgrade to a combined simulation/deployment system. Of course, there are problems with this approach, including the need to learn and combine two different configuration methodologies, and a possibly-large performance penalty. We plan to address these problems as part of our work on this proposal.

Our goals include:

- **Complete integration.** A user should be able to divide responsibility between *ns* and Click in whatever way makes the most sense for their application. *Ns* might be useful for traffic generation, or maybe Click has exactly the right elements to generate the relevant protocol; the user shouldn't be forced to choose one or the other because of secondary concerns.
- **Fast simulation.** Click-plus-*ns* should not perform horrendously slower than *ns* alone.
- **Usability.** The system should make the join between the two systems as seamless as possible. For example, a single file should describe the combined Click-plus-*ns* configuration.

The combined-*ns*-and-Click model is actually two models, distinguished by which system is embedded in the other. Part of our initial work will be to evaluate these models, and decide which ones merit extended development. However, the differences between them are less important than their similarities. Both have the same advantages and disadvantages relative to Click-only simulation; and both are better than *ns*-only simulation, because both of them integrate simulation and deployment.

5.1 Embedding Click in *ns*

A simple *ns* configuration consists of nodes connected by links. Applications and agents, which represent things like traffic sources, attach to the nodes. *Ns* nodes generally take packets arriving over links and demultiplex them among the relevant agents. One natural approach to *ns*-Click integration would be simply to support a Click agent type. Packets handed to such an agent would be passed to Click, and processed through some configuration graph. To take this idea even further, one could support Click nodes, where packets arriving at the node avoided *ns*'s node implementation and went straight to a Click graph.

Recent work at the University of Colorado has done exactly this [20]. Ns2click, which has not yet been released, lets users attach Click routers to *ns* nodes. These attached routers become instantly integrated with *ns*, able to accept, process, and route *ns* packets arbitrarily. Ns2click simulations can run significantly slower than *ns* simulations; this might be avoidable, or it might be due to overhead inherent in translating between *ns* and Click's packet formats, for example.

Ns2click arose out of a project that built several deployable wireless routing protocols using reusable Click elements. The authors wanted to compare their implementations against the best reference implementations available, in the *ns* simulator; ns2click was the result. Dirk Grunwald, Robert Morris, and two of us (Karp and Kohler) recently submitted an NSF ANR/Special Projects proposal (*An Infrastructure for Rapid Deployment and Experimentation for Ad Hoc Networks*, in process) to extend this work into a full-fledged ad hoc routing protocol toolkit. The simulation-to-deployment story described there has motivated this expanded proposal. Ns2click forms a minor part of the prior proposal; in particular, the proposal suggests some speed improvements, and investigating other simulators like GloMoSim [28]. However, much more exploration and research must be done to create a general Click-in-*ns* simulator, which is what we propose to do here. For example:

- For some applications, such as wireless routing, Click should take over an entire *ns* node. For others, however, such as protocol design or wired routing protocols, Click might make more sense as an agent or application, coexisting with other *ns* agents and applications.
- Click-in-*ns* nodes, like normal *ns* nodes, should support communication to and from *ns* agents and applications.
- Ns2click lets *ns* store packets in its own queues, rather than Click's. This prevents experimentation with queueing, scheduling, and active queue management. Click-in-*ns* should let Click optionally handle packet storage, or at least let Click manipulate the *ns* queues.

5.2 Embedding *ns* in Click

An obvious alternative is to switch the direction and embed *ns* in Click. In ns2click (that is, Click-in-*ns*), the *ns* component is in charge of the simulation, constructing the topology and determining what events happen when. Click is called only when *ns* determines it is necessary. In *ns*-in-Click, the Click component would be in charge. *Ns* functionality would be encapsulated in elements—an "*NsTCPAgent*", for example—and Click would call out to these elements to use *ns* functionality.

The choice between the two models, Click-in-*ns* and *ns*-in-Click, depends on the simulation task at hand. When testing a large topology with a couple of Click nodes, Click-in-*ns* makes more sense; but when testing a single network component with *ns* traffic generators, *ns*-in-Click fits better. Of course, if a particular Click topology doesn't fit into *ns*'s application–agent–node model, only *ns*-in-Click will do.

The most natural integration of *ns* into Click will probably be at the application and agent level. An *ns* agent, which represents a single protocol, naturally corresponds to a Click element. Packets arriving at the element would be forwarded to the agent, just as an *ns* node might do; and, likewise, packets sent by the agent would be emitted by the element. These encapsulated *ns* agents would make great traffic generators.

5.3 Discussion

Like any integration effort, combining *ns* and Click will present development challenges. We believe that only one is both fundamental and difficult: simulation speed. For example, a packet that transfers between *ns* and Click multiple times may require a full data copy at each transfer point, since *ns* and Click have different implementations of the packet concept. We will measure this overhead, and other similar overheads, and determine whether *ns*-plus-Click demands any serious performance work. If it does, there are several possible solutions. One could build a special *ns* driver for Click, tailoring Click's abstractions to *ns*'s format (some ns2click work might be relevant here), or one could modify *ns* to opaquely carry the information Click requires (something like this was done for *ns* emulation).

All the improvements described in Section 4.5 would also benefit a mixed Click-and-*ns* approach, and we plan to implement them regardless of the approach we choose. The combined approach also offers some new usability challenges: How can we ease the transitions between *ns* and Click syntax and semantics?

6 Applications

Creation of the software infrastructure we propose is a significant step toward turning network simulator code into deployable research network systems. Should this proposal be funded, we will apply these tools we create to build systems that solve problems in active queue management and wireless networking, both of which are investigated nearly exclusively in simulation today, without the creation of credible deployable implementations; and in network debugging and testing tools, pursued today in deployment alone, without the benefit of a controlled simulated testing environment. In this way, we contribute further to the advancement of deployable network systems, and can measure our degree of success at building a useful and relevant infrastructure.

6.1 Router-based congestion control mechanisms

As discussed in Section 1.1, we are currently involved in the simulation-based comparisons of active queue management mechanisms, and plan to use the tools developed in this project to allow the comparison of active queue management mechanisms in testbeds and in actual networks as well as in a simulator.

In addition, we plan to develop tools to allow ourselves and other researchers to have some of the same capabilities with other router-based mechanisms for congestion control currently investigated mostly in simulations. In particular, we believe that research on router mechanisms for detecting and controlling misbehaving flows and on detecting and controlling aggregates in times of high congestion would particularly benefit from an easier migration path from simulations to experiments and deployment.

6.2 Wireless networks

The vast majority of wireless networking research, especially in ad-hoc routing and TCP performance over wireless networks, has been pursued in simulation, without construction of deployable systems. The expense of deploying a large-scale wireless infrastructure has thus far kept experimentation out of reach of researchers, and thus reinforced the trend away from building deployable versions of experimental wireless network software systems. In the NSF ANR/Special Projects proposal *An Infrastructure for Rapid Deployment and Experimentation for Ad Hoc Networks* (under review), Karp and Kohler, with Robert Morris of MIT and Dirk Grunwald of the University of Colorado, plan to extend Click into a toolkit for building ad-hoc routing protocol implementations. We now propose further to use the set of varied ad-hoc routing protocol implementations that result in a comprehensive comparison on even footing, in the same Click simulator. The results of such a comparison will help bring clarity to a field in which published simulation results have often been incomparable, because of different simulator versions and different simulation parameters.

In the few deployment studies of ad-hoc networks to date, measurements of such quantities as link error rates, node positions over time, transport protocol behavior, and routing protocol behavior have been difficult to take and analyze. As an example, the ACTCOMM project at Dartmouth and other institutions [1] (in which Brad Karp collaborated between 1997-1999) made repeated demonstrations of an ad-hoc routing testbed involving around twenty mobile users with radio-equipped laptops. The measurement infrastructure used in that project involved a haphazard web of scheduled shell scripts, to read diverse statistics counters and logs maintained locally all throughout the code of the running demo system. We will use the modular measurement methodology described in Section 4 to allow comprehensive, easily controlled, and therefore more easily analyzed instrumentation of wireless network system deployments.

Our (Karp's and Floyd's) ongoing work in improving DSACK TCP's performance on wireless (and wired) networks that reorder packets, and produce skewed round-trip time distributions because of link-layer retransmission, is a natural domain in which to apply the combined simulation and deployment infrastructure we propose. To date, our simulations have used simple probabilistic models for generating packet reorderings and link errors. There is little known about how to model these phenomena accurately in simulation. We will deploy our algorithmic improvements to TCP in Click on wireless networks, and learn how our TCP enhancements behave in the real wireless environment.

6.3 Measurement: network testing and debugging tools

Measurement, as a pivotal tool for testing and debugging new network systems under development, and for assessing the poorly understood large-scale behavior of deployed network systems, has played a central role in the evolution of the Internet. The most broadly useful measurement systems (*e.g.,* libpcap, tcpdump, traceroute, pathchar, and tcpanaly) were all built significantly after the deployment of the protocols they measure. We believe that *co-design* of network systems with measurement and testing code that all operate both in simulation and on a production network will significantly ease the deployment of future research network systems.

The most traditional use of measurements in the community is to quantify the performance of network systems, motivating and evaluating improvements to the designs of those systems. But there is a much broader virtuous circle we will exploit, that links real-world deployment of a network system, controlled simulation of that system, and measurements of the deployed and simulated versions of the system. We claim that among these three, detailed knowledge of any two greatly benefits the third.

For example, consider building a Click TCP implementation. In parallel with the TCP itself, we will simultaneously build an active measurement tool, also in Click, for provoking the range of behaviors TCP should exhibit. This measurement tool will validate our Click TCP, but will *also* validate deployed TCP implemenations over the real Internet. As we develop the Click TCP, comparing these measurements will allow us to say with confidence that the Click TCP behaves as do a particular set of already deployed TCPs. Here, measurements of the deployed system help us bootstrap a new protocol implementation.

But our ability to observe the workings of the Click TCP aids us in developing the measurement tool, as well! The development cycle for Internet measurement tools today begins with deployment on the real Internet, where the internal state of the entities being measured is typically unavailable for examination.

When measurement code itself behaves unexpectedly, disambiguating a bug in the measurement code from a bug in the system being measured is a time-consuming process. Our use of the *same* measurement code in a controlled Click simulator, where the state of the measured entities is visible, will ease and speed measurement system development.

Our experience in the development of a TCP measurement tool for use on the Internet leads us to these conclusions. We (Padhye and Floyd) developed TBIT, the TCP Behavior Inference Tool [22], to characterize and test TCP implementations—specifically, the algorithm variations they employ and the parameters they use. The development of TBIT would have been significantly eased by implementation in a Click for dual simulation and deployment use. These advantages clearly will extend to co-development of other network systems and code to measure them. If this proposal is funded, we will build a TBIT in Click, and extend the Click TBIT to consider a wider range of TCP behaviors—including the extensive TCP validation tests, and those for other protocols, as appropriate, that are part of today's *ns*, thus letting us apply them to real-world deployments, and not only simulations. We will also build a suite of measurement tools for other network systems, including AQM schemes and ad-hoc routing schemes for wireless networks.

7 Relation to ongoing work

Sally Floyd is one of the two co-founders of the *ns* simulator, and she has been continuously involved with its development. She also has a long history of interest in the validation of protocols, both for simulations and for implementations deployed in the Internet. Her other ongoing work has mostly been on the understanding and development of transport protocols, and on router mechanisms for class-based queueing, active queue management, explicit congestion notification, detecting and policing misbehaving flows, and aggregate-based congestion control. The further development of all of this work will benefit from the ability to more easily bridge the gap between simulations, experiments, and deployment. All of her work for the last three years has been supported by funding from AT&T.

Eddie Kohler developed Click for his doctoral work at MIT. He continues to focus on Click as a networking system, as a community resource, and as an opportunity for systems-oriented programming language research. His ongoing work on Click's development is supported by the Xorp project at the ICSI Center for Open Networking (ICON, funded by Intel and AT&T/ACIRI), and by Mazu Networks, Inc., a startup at which he is Chief Scientist. Xorp, the eXtensible Open Routing Platform, is building routers that are simultaneously designed for research, and fast and stable enough for production use. Xorp routers will use Click as their forwarding paths, and can clearly benefit from better simulation capabilities.

Dr. Karp designed and evaluated the GPSR geographic ad-hoc routing algorithm in his doctoral work at Harvard. He also built a distributed implementation of a loop-free ad-hoc routing system for laptops at Harvard. At ICSI, Dr. Karp continues to work on geographic routing and ad-hoc networks, where he pursues projects with three students investigating geographic traffic engineering for ad-hoc networks (to route around congestion geographically); and enhancements to the TCP protocol to improve its robustness on ad-hoc networks where the round-trip time on a network path is highly variable, and packet reordering may occur because of route changes.

Jitendra Padhye has worked in the areas of TCP and TCP-friendly congestion control, and his thesis was on a model-based approach to TCP-friendly congestion control. His work has included simulations, analysis, and the development of tools for the active measurement of TCP behaviors in the Internet. One of his current projects is the simulation-based comparison of various proposals for active queue management, as discussed in this proposal.

8 Technological and societal impact

We believe that the current uncontrolled divergence between simulation and deployment in networking research has already had adverse impact on the Internet at large. As one likely example, explicit congestion notification promises to make the current Internet better for everyone, by reducing the impact of congestion events on everyday data transfers; yet it has not been widely deployed, partially because of network operators' skepticism for AQM and other systems tested mainly in simulation. The flip side is that bridging the simulation–deployment gap could make the Internet better. Researchers will easily be able to move their inventions from the hothouse of simulation into availability for society at large. This will facilitate the development of new applications, the deployment of new technologies, and the evolution of the Internet at large.

We are also committed to improving the collective state of network research—to helping our community. Our goal is to provide well-packaged tools that make it easier both to conduct good research, and to learn what good research is. As such, we will focus on communication and education: on explaining what we have done, and on making it easy to use.

9 Research plan

This section summarizes our current plan for how we will accomplish the goals of the proposal; please refer back to earlier sections for more information on the particular work items. This plan is extremely flexible, of course, and the actual order of execution depends on factors unknown to us now, such as the preferences of particular students that express interest in working with us.

- **Year 1: Evaluation.** Investigate *ns* and Click in more detail. Build prototypes of *ns*-in-Click and Click-in-*ns* simulators, and evaluate which approach is more promising. Begin implementing more complex Click traffic generation elements. Research usability improvements for large Click configurations, looking particularly at programming language techniques.

- **Year 2: Initial implementation.** Combine Click and *ns* as seamlessly as possible. Investigate active testing and debugging tools. Continue adding Click simulation elements. Add initial programming language improvements. Research dynamic configurations.

- **Year 3: Solidity.** Investigate Click and Click-plus-*ns*'s scaling properties (how large a simulation can you run?). Implement performance enhancements as required. Implement initial applications (Section 6). Continue adding Click simulation elements. Use testing and debugging tools to test those elements.

- **Year 4: Usability.** Initial release of Click-plus-*ns* toolkit. Use release to investigate the usability of the combined system. Add programming language tools for analyzing and transforming simulation configurations. Continue adding Click simulation elements. Continued application development, with feedback into the toolkit.

- **Year 5: Deployment and documentation.** Continued toolkit solidity improvements. Toolkit documentation. Development of community mechanisms for sharing simulation configurations and elements. Integration with deployment infrastructures like Emulab. Complete application development.

Part D
REFERENCES CITED

[1] ACTCOMM project. The ACTCOMM transportable agents for wireless networks project (Web site). `http://actcomm.dartmouth.edu/`.

[2] Sanjeewa Athuraliya, Steven Low, and David Lapsley. Random Early Marking. In *Proc. 1st International Workshop on Quality of Future Internet Services (QofIS 2000)*, September 2000.

[3] Sandeep Bajaj, Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, Padma Haldar, Mark Handley, Ahmed Helmy, John Heidemann, Polly Huang, Satish Kumar, Steven McCanne, Reza Rejaie, Puneet Sharma, Kannan Varadhan, Ya Xu, Haobo Yu, and Daniel Zappala. Improving simulation for network research. USC computer science department technical report 99-702b, University of Southern California, September 1999.

[4] Lawrence S. Brakmo and Larry Peterson. Experiences with network simulation. In *Proc. 1996 ACM SIGMETRICS Conference on Measurement and Modeling of Computer Systems*, pages 80–90, May 1996.

[5] Mikkel Christiansen, Kevin Jeffay, David Ott, and F. Donelson Smith. Tuning RED for Web traffic. In *Proc. ACM SIGCOMM 2000 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, August 2000.

[6] Kevin Fall. Network emulation with the NS simulator, October 1998. `http://www.isi.edu/nsnam/ns/ns-emulation.html`.

[7] W. Feng, D. Kandlur, D. Saha, and K. Shin. Blue: A new class of active queue management algorithms. Technical Report CSE-TR-387-99, University of Michigan, 1999.

[8] Sally Floyd, Ramakrishna Gummadi, and Scott Shenker. Adaptive RED: An algorithm for increasing the robustness of RED's active queue management, August 2001. `http://www.aciri.org/floyd/papers/adaptiveRed.pdf`, submitted for publication.

[9] Sally Floyd and Van Jacobson. Random early detection gateways for congestion avoidance. *IEEE/ACM Transactions on Networking*, 1(4):397–413, August 1993.

[10] Flux Research Group. Emulab.net: The Utah network testbed (Web site), 2001. `http://www.emulab.net/`.

[11] C. Hollot, Vishal Misra, Don Towsley, and Wei Bo Gong. On designing improved controllers for AQM routers supporting TCP flows. In *Proc. IEEE INFOCOM 2001*, April 2001.

[12] Norman C. Hutchinson and Larry L. Peterson. The *x*-kernel: An architecture for implementing network protocols. *IEEE Transactions on Software Engineering*, 17(1):64–76, January 1991.

[13] Gianluca Iannaccone, Christof Brandauer, Thomas Ziegler, Christophe Diot, Serge Fdida, and Martin May. Comparison of tail drop and active queue management performance for bulk-data and Web-like Internet traffic. In *Proc. 6th IEEE Symposium on Computers and Communications*, July 2001.

[14] Gianluca Iannaccone, Martin May, and Christophe Diot. Aggregate traffic performance with active queue management and drop from tail. *Computer Communication Review*, 31(3), July 2001.

[15] NIST Internetworking Technologies Group. NIST Net home page (Web site), 2001. `http://www.antd.nist.gov/itg/nistnet/`.

[16] Eddie Kohler, Robert Morris, Benjie Chen, John Jannotti, and M. Frans Kaashoek. The Click modular router. *ACM Transactions on Computer Systems*, 18(3):263–297, August 2000.

[17] Srisankar Kunniyur and R. Srikant. Analysis and design of an adaptive virtual queue (AVQ) algorithm for active queue management. In *Proc. ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, August 2001.

[18] Dong Lin and Robert Morris. Dynamics of random early detection. In *Proc. ACM SIGCOMM '97 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 127–137, October 1997.

[19] Robert Morris. Scalable TCP congestion control. In *Proc. IEEE INFOCOM 2000*, pages 1176–1183, March 2000.

[20] Mike Neufeld, Audun Tornquist, and Dirk Grunwald. NS2Click (Web site), 2001. `http://systems.cs.colorado.edu/Networking/ns2click.html`.

[21] *ns* project. The network simulator—ns-2 (Web site). `http://www.isi.edu/nsnam/ns/`.

[22] Jitendra Padhye and Sally Floyd. On inferring TCP behavior. In *Proc. ACM SIGCOMM 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pages 287–298, August 2001.

[23] Vern Paxson and Sally Floyd. Why we don't know how to simulate the Internet. In *Proc. 1997 Winter Simulation Conference*, December 1997.

[24] K. K. Ramakrishnan, Sally Floyd, and David Black. The addition of explicit congestion notification (ECN) to IP. RFC 3168, Internet Engineering Task Force, September 2001. `ftp://ftp.ietf.org/rfc/rfc3168.txt`.

[25] Luigi Rizzo. Dummynet: a simple approach to the evaluation of network protocols. *Computer Communication Review*, 27(1):31–41, January 1997.

[26] Curtis Villamizar and Chuck Song. High performance TCP in ANSNET. *Computer Communication Review*, 24(5), October 1994.

[27] *x*-kernel project. The *x*-kernel protocol framework (Web site). `http://www.cs.princeton.edu/xkernel/`.

[28] Xiang Zeng, Rajive Bagrodia, and Mario Gerla. GloMoSim: A library for parallel simulation of large-scale wireless networks. In *Proc. Workshop on Parallel and Distributed Simulation*, pages 154–161, 1998.