Capturing Domain Knowledge Through Extensible Components

Erik Kline, Genevieve Bartlett(✉), Geoff Lawler, Robert Story, and Michael Elkins

Information Sciences Institute, University of Southern California, Los Angeles, USA
{ekline,bartlett,glawler,rstory,elkins}@isi.edu
http://www.isi.edu

Abstract. Recreating real-world network scenarios on testbeds is common in validating security solutions, but modeling networks correctly requires a good deal of expertise in multiple domains. A testbed user must understand the solution being validated, the real-world deployment environments, in addition to understanding what features in these environments matter and how to model these features correctly in a testbed. As real-world scenarios and the security solutions we design become more diverse and complex, it becomes less likely that the testbed user is able to be a domain expert in their technology, a field expert in the deploy environments for their technology, and an expert in how to model these environments on the testbed. Without the proper expertise from multiple domains, testbed users produce overly simplified and inappropriate test environments, which do not provide adequate validation. To address this pressing need to share domain knowledge in the testbed community, we introduce our Extensible Components Framework for testbed network modeling. Our framework enables multiple experts to contribute to a complex network model without needing to explicitly collaborate or translate between domains. The fundamental goal of our Extensible Components is to capture the knowledge of domain experts and turn this knowledge into off-the-shelf models that end-users can easily utilize as first-class testbed objects. We demonstrate the design and use of our Extensible Components Framework through implementing Click Modular Router [10] based Extensible Components on the DETER testbed, and advocate that our framework can be applied to other environments. We focus on *wired* network models, but outline how Extensible Components can be used to model other types of networks such as wireless. (This material is based on research sponsored by DARPA under agreement number HR0011-15-C-0096. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation thereon. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA or the U.S. Government.)

H. Gao et al. (Eds.): TridentCom 2018, LNICST 270, pp. 141–156, 2019.
https://doi.org/10.1007/978-3-030-12971-2_9

Keywords: Testbed validation · Network models · Knowledge capture · Network testbed · Testbed experimentation

1 Introduction

Validation of security and networking solutions requires creating and harnessing complex scenarios and realizing those scenarios on a testbed infrastructure. A testbed user must decide what aspects of a real-world environment play a role in validation and determine how to map these real environment features into the testbed environment correctly. Proper validation requires a deep understanding and meaningful expertise across multiple domains [7]. Specifically, to perform proper validation with today's technology, a testbed user must maintain expertise across three distinct domains: First, a testbed user must understand the solution being validated. Second, she must understand the real-world environments where the solution will be deployed. Last, the user must understand what features are relevant in these environments and how to model these features correctly in a testbed. As the real-world has grown increasingly more diverse and complex, the burden on the testbed user has grown immensely. It has become less and less likely that a testbed user can maintain being a domain expert in her technology, a field expert in the deploy environments for her technology, and an expert in how to model these environments on the testbed. In lieu of working with field and modeling experts, testbed users often produce overly simplified test environments, or use canned modeling approaches unsuitable for their experiment which do not provide adequate realism for testing and validation.

Collaborating with field and modeling experts reduces the burden on the testbed user while working towards more realistic testing scenarios, but such a collaboration requires a good deal of effort to translate expertise between domains. Ad-hoc translation between domains can be tedious and without structured *knowledge capture*, the time spent sharing expertise to build testbed models has little pay-off beyond the initial effort. Low-effort knowledge capture to facilitate reuse, sharing and repeatability is thus a common goal in testbed validation.

In this paper, we describe our work to perform knowledge capture and domain expertise translation in testbed network modeling for experimental validation. We achieve this knowledge capture through a framework we call the *Extensible Components Framework*. A key feature of our Extensible Components is the capture and dissemination of knowledge at three distinct levels—the model approach, the model specification and how to composed this model into a larger network model. Further, we treat Extensible Components as first-class testbed objects. That is, the testbed understands what an Extensible Component is and how to allocate and instantiate one given an experiment configuration. First-class status for Extensible Component allows experimenters to easily deploy and use complex models and scenarios without having to gain significant knowledge on the components implementation, modeling approach or configuration.

We describe the three expert domains our Extensible Components Framework brings together through knowledge capture (Sect. 2). We then discuss how

to: create an Extensible Component (Sect. 3.1), extend the component by specifying resident models (Sect. 3.2), and how end-users easily use, combine and share these models (Sect. 3.3). We follow with example components of a wired network (Sect. 4.1) and wireless network (Sect. 4.2) and include a brief discussion on our experiences with users and experts using our Extensible Components. We close by discussing related work and how our Extensible Components Framework can incorporate many existing tools. Our initial implementation and use of our framework is freely available and currently implemented on the DETER testbed [11]. We discuss the straightforward process of porting to other testbed platforms in Sect. 3.4.

2 Extensible Components

The fundamental goal of our Extensible Components Framework is to be able to capture the knowledge of domain experts and turn this knowledge into off-the-shelf models that end-users can easily utilize.

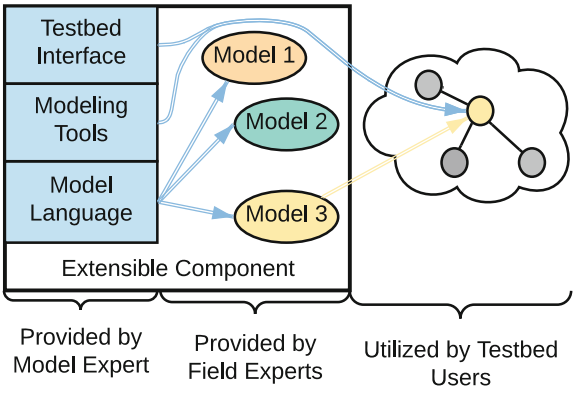

Fig. 1. Extensible component overview

We capture and disseminate domain expertise from three distinct experts—Modeling Experts, Field Experts and Users. A Modeling Expert builds tools that accurately model a specific domain, providing the *ability to model* their domain on a testbed to others. This ability to model is the core of an Extensible Component. Field Experts then extend an Extensible Component by building multiple specific models of their domain. Finally, users compose these specific models to build aggregate networking models which they can experiment upon, reuse and share with others. An overview of how this information is captured, provided and shared is illustrated in Fig. 1 while the propagation of information is shown in Fig. 2.

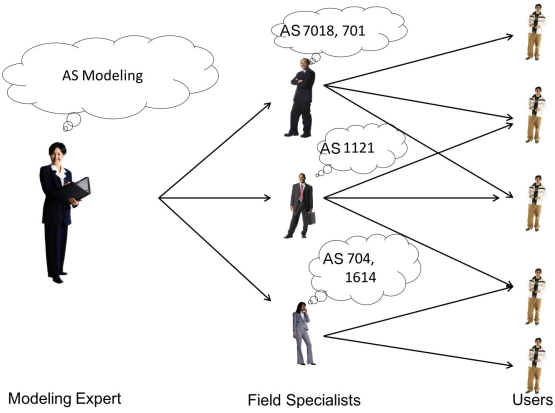

Fig. 2. The three levels of knowledge capture and how knowledge propagates

2.1 Modeling Experts and Languages

The critical role of the Modeling Expert is to be able to provide others with the ability to model their specific domain. Importantly, they understand the techniques required to accurately model their domain within a set of given constraints while recognizing the artifacts these techniques may create. For example, a Modeling Expert specializing in backbone networks understands what features are important in modeling high capacity links, exchanges and routers, and understands the limitations of his modeling approach and tools used to realize this approach.

Different domains require different modeling techniques. A Backbone Modeling Expert is focused on modeling high-capacity wired networks and understands modeling features such as bandwidth and delay enforcement, queuing strategies and routing techniques. A Wireless Modeling Expert requires knowledge of different features, such as understanding the properties of wireless channel access strategies, wireless collisions, and hidden nodes. The Backbone Modeling Expert does not care about nor know how to model the concerns of the Wireless Expert and vice versa.

Importantly, multiple Modeling Experts and modeling approaches can exist for the same domain. Different model approaches make different trade-offs between resources and fidelity, modeling certain features with high-fidelity and reducing fidelity in other aspects. For example, one could model Autonomous System (AS) networks using a link-centric approach or a router-centric approach. A link-centric approach provides high fidelity modeling of link properties and congestion while providing relaxed fidelity for modeling of routers and routing protocols. Conversely, a router-centric approach offers a high fidelity model of routers and routing protocols while abstracting many of the link properties, which may be more appropriate than a link-centric approach for testing new routing protocols. Both of these experts are modeling ASes but their approaches provide different trade-offs for experimenters. Supporting diversity in modeling

approaches enables broader applicability, as not all experiments require the same level of fidelity in all features.

The Modeling Expert can also provide controls on the amount of information that may be visible through her modeling tools. An example of this would be a model of an AS which does not leak any topological information to the edge users, similar to the current Internet. This ability to provide privacy is important for Field Experts who want to create models for others to test against, but do not want to expose any proprietary information about their structure. Conversely, a Modeling Expert may not provide these guarantees where they cannot be enforced or do not make sense to enforce. For example, a model approach for an AS may have to expose topological information when testing new routing protocols in order make validation guarantees. In both cases, the Modeling Expert can pass down this knowledge of constraints to Field Experts to ensure Field Experts and Users understand the privacy and fidelity guarantees.

Beyond providing tools and an approach to properly model a domain, Modeling Experts must also provide an easy way for others to customize and create specific models using their tools and modeling approach. For this, we advocate adoption of small languages that can easily and accurately describe and set the features of a model. Using a language to describe a model, rather than parameterizing a blob of code, provides flexibility and scaling, such as using loops to create and set multiple parts of a model. For example, an AS model may be described with a simple edge-list link language that provides an easy description of a network and its link properties. Similarly, a wireless model may utilize a node-list approach that describes the position of speakers, their mobility and their transmission capabilities. Whatever language is chosen, the goal is for the language to easily describe a model, without requiring the person describing the specific model to understand the underlying model implementation or even the modeling approach—beyond just a broad understanding its goals and limitations.

Finally, to provide others with a general idea of a component's applicability, Modeling Experts need to provide some documentation. This documentation should give a brief and high-level description of the overall modeling approach, and the capabilities and constraints of the model, allowing Field Experts and Users to make informed decisions when choosing a component.

2.2 Models and Field Experts

After a Modeling Expert provides the ability to create and exercise a model, it falls upon Field Experts to extend components by generating models. Field Experts have significant knowledge in their relevant field but may not have knowledge of *how* to model their domain. For example, an operator of an AS knows the properties of their links and structure of their topology but may not know how to accurately represent that on a testbed. Instead of learning how to model, the Field Experts simply utilize the languages provided by a Modeling Expert to describe their system and generate a model of it on the testbed.

Of course, Modeling Experts and Field Experts are not mutually exclusive. Often a Modeling Expert will have significant knowledge about one or more fields, and a Field Expert may know how to approach testbed modeling. However, we expect that there are significantly more Field Experts than Modeling Experts, as accurately modeling systems is a difficult challenge and a niche area of expertise. Thus, it makes sense to capture and amplify a Modeling Expert's knowledge.

Just as Users can compose models into a larger network model, Field Experts can also draw on multiple Extensible Components and are not required to use only one Modeling Expert's approach. For example, an AS Field Expert may want to represent her system using multiple models—a link-centric model, a router-centric model and a hybrid model. As long as the Field Expert can understand the trade-offs, as accurately described by the Modeling Expert, she can make rational and intelligent decisions about which Extensible Components to use.

Ultimately, the goal of the Field Expert is to generate models through the Modeling Experts tools, and provide those models to users who can run experiments using these models. It is incumbent on the Field Experts to select the Extensible Components they feel adequately describe their domains while also meeting their privacy concerns as described in Sect. 2.1. Further access controls for these models can be provided and enforced by the testbed.

2.3 Users

Users are the final level of our three-level approach, and are the ultimate recipient of the captured knowledge in our Extensible Components Framework. The fundamental goal of extensible components is to allow experimenters to utilize complex network systems within their experiments without having to obtain domain knowledge of a specific field or modeling approach. Building on knowledge from Modeling and Field Experts, the User can now simply grab one or more of the provided models and begin to experiment against them. The User does not need to understand how the model is implemented, or the structure of the model itself, in order to conduct an experiment. All the User needs to know is the goal of a model and what constraints and artifacts may occur by utilizing this model. With this information, the User can conduct complex and scientifically valid experiments without spending the tedious time becoming a domain expert in modeling or deployment environments.

3 Component Lifecycle

In the previous section we described how an Extensible Component brings together expertise from Modeling and Field Experts to unburden the testbed user and promote reuse. Next we describe the details of how a component is created and used. We have developed and used our Extensible Component prototypes on the DETER testbed [11], but believe porting our components to other testbeds would be a straightforward process.

3.1 Creating a Component

In order to create a new component, the Modeling Expert is required to provide three building blocks—a small language, a tool or set of tools to realize the component, and an interface to generate and exercise the component. As discussed in Sect. 2.1, the language is utilized by Field Experts to describe a model. We provide no restrictions on the choice of language, providing flexibility to the Modeling Expert to choose whatever best suits their tools. The expert is free to utilize languages that already exist if that is the best option.

The tool created by the Modeling Expert receives models specified in the component's language, and instantiates a functional component representing that model as output. The choice of tool is once again up to the Modeling Expert. Testbeds, such as DETER, can provide canned capabilities, such that the expert may only need to translate their language into one that a canned capability already understands. For example, one of the components we've created for DETER uses the Click Modular Router [10] as it's underlying implementation. Thus, our component tool translates our modeling language into a configuration file that Click understands.

The final building block needed to be able to create and utilize a new component is an interface to the testbed that allows the component to be allocated and instantiated as a first-class object. For DETER, this is a TCL (NS) interface. We provide Modeling Experts with a TCL library that they can import and build off of to provide their interface. These experts simply need to sub-class the component class and then implement the abstracted functions. These functions are 'create' and 'add-link' which indicate to the testbed how many nodes to allocate to the component and how to connect other testbed objects to this component. Further, the component class accepts an arbitrary number of arguments to be passed to the sub-class, allowing Modeling Experts to accept variability during the component construction.

```
set myc [$ns component ASWAN AS701]
```

Fig. 3. Example component instantiation

Figure 3 shows the basic Extensible Component API as supplied by the component library. This example tells DETER to allocate a new component which is of type ASWAN (a component for modeling an Autonomous System Wide Area Networks). Everything else will be passed to the component creation method to be handled by the specific component sub-class. Thus, in this example, our Modeling Expert created a component called ASWAN, which expects a single argument, the specific model to instantiate. The specific model, AS701, is created and named by the Field Expert (and models a specific Autonomous System's network environment).

3.2 Creating a Model

The complexity of creating a new component allows for the rest of the framework to be significantly simpler. To create a model, the Field Expert simply needs to pick a component and utilize the given language to express the model. The challenge for the Field Expert is picking a component that can adequately express their model while also understanding the components capabilities, constraints and potential artifacts. For example, a component that models ASes in a link-centric fashion may have a maximum number of links per CPU it can model before some artifacting occurs. The Modeling Expert documents what these constraints and limits, and the Field Expert must heed the documentation to create useful and accurate models.

```
11 12 {"bw": "400Mbps",
       "delay": "10ms",
       "loss": "0.0001"}
11 13 {"bw": "200Mbps",
       "delay": "15ms",
       "loss": "0.0001"}
```

Fig. 4. Small example of a modeling language

Once the component has been chosen, the expert creates their model within the Extensible Component's language. The code segment in Fig. 4 is an example of a link-centric modeling language using an edge list format. The Field Expert has used this language to describe a modeled network with three routers and two edges (links "11"–"12" and "11"–"13"), with several link constraints specified for each edge. Once the model has been specified, it is imported into an archive or library. On DETER, the models are given a name and stored in a model repository for later use.

3.3 Using a Component

Utilizing an Extensible Component within an experiment is where everything comes together. First, the interface created by the Modeling Expert is exercised by the user to define, instantiate and deploy the component. On DETER, this involves importing the TCL library and defining the component within the topology as shown in Fig. 3. The user can pass any options to the component through this interface. Further, the user can connect the component to other experiment objects using the 'add-link' API, which exposes specific I/O portions of the component allowing the component to be physically linked to other experimental hardware. At this point, the user does not need to do anything additionally to have their component realized. When the experiment is instantiated, the component will be automatically configured, initialized and enabled.

3.4 Deploying Extensible Components on a Testbed

The majority of our discussion of Extensible Components has been based on our deployment on the DETER testbed [11]. However, the overall framework should be amenable to most testing infrastructures. To utilize our Extensible Components Framework on any given testbed, three fundamental capabilities are required. First, the infrastructure must be able to express a component as a first-class object like any other object in that testbed. This means the testbed back-end will deploy, connect and configure the relevant infrastructure to instantiate the component. Second, a user must be able to express the component within their experiment configuration. On DETER, this is accomplished via a TCL (NS) library available to any user and other testbeds would require similar modifications or extensions. Finally, a repository to store Extensible Components and models generated by Modeling Experts and Field Experts, respectively, is required so that the Extensible Components and models can be easily retrieved and used by experimenters.

4 Experiences with Example Components

We have two example components in use that demonstrate the domain expertise separation of our Extensible Components Framework and the capabilities of Extensible Components. Both of these Extensible Components have been used to make multiple models of real-world, toy example and corner-case testing models.

The first component, Wide area network (WAN) emulation, is freely available and has been used by multiple teams for testing and evaluation in a recent DARPA program. The second component, wireless network emulation, is currently in development for a second DARPA program. Both of these components currently use the same Click-based [10] back-end for realization. We are working towards augmenting Click with additional tools to realize these components, as other technologies are better suited to accurately model specific elements in each domain. A key feature of our Extensible Components Framework is the separation of the component use from the model implementation. As we change the underlying tools used to realize a model, the testbed user interface and configuration interface of the model can remain largely the same, reducing the burden on testbed users and Field Experts.

4.1 Wide Area Network Emulation

Wide area network (WAN) emulation is a common need for testbed validation, and typically requires accurate and realistic topologies. In over nearly a decade of working with DETER, we observed that without easily accessible and usable network emulation, experimenters utilize excessive testbed resources simulating real-world topology. This network simulation approach is not an efficient use of resources as each physical machine used for simulation typically acts as a simple store and forward router, and provides nothing else to the experiment.

We needed a technology that would allow experimenters to utilize the domain knowledge of WAN operators without having to gain that knowledge themselves, while also being far more efficient in resource allocation.

To address this inefficient use of resources, we implemented a WAN Emulation Extensible Component. The WAN emulation component allows an experimenter to deploy an emulated WAN of multiple routers and links on a small number of physical resources (usually just one physical machine, but in complex topologies, this is expanded to more).

As researchers on modeling and testbeds, we were the Modeling Experts for the WAN Emulation Extensible Component. We created all three of the essential building blocks required in our Extensible Components Framework (as discussed in Sect. 3.1). Our component specifies an edge-list based modeling language, similar to that seen in Fig. 4, which allows Field Experts to describe routers and the links between them across multiple features. These include bandwidth, delay, multiple loss modeling approaches, multiple queuing strategies, AQM, packet reordering, random bit error as well as many others. The second building block is a tool that converts the model as described in the modeling language into a format that can be utilized by the testbed. To build this tool, we leveraged the existing Click [10] capabilities of DETER. Thus, our python-based tool simply translates our edge list into a Click configuration. This tool is exercised at experiment run-time, pulling the specified model from the model repository, translating it and instantiating Click.

Finally, we built additional TCL interfaces to allow users to exercise the component as a first-class testbed object. This interface is very simple, requiring the user to only specify the model they wish to instantiate. Further, the user is required to connect the models I/O interfaces to other network components. An example connection is shown in Fig. 6 which connects an experimental node called "mynode" to our component at an I/O location called "router1". This means that a physical link between mynode and our component will be constructed and the data from that link will enter the component's topology at router1.

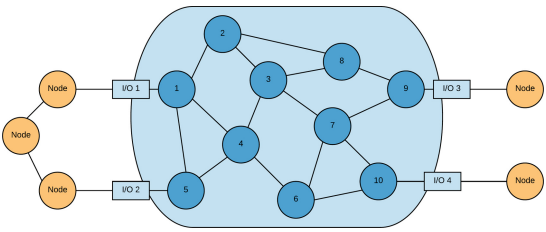

Fig. 5. Logical topology of our WAN with component in blue and other objects in orange. The square boxes represent I/O points where data enters and exits the component. (Color figure online)

An example testbed topology linking into a model from our WAN emulation component can be seen in Fig. 5. The highlighted blue area represents a WAN topology model. The four squares at the edge of this zone represent the I/O points where the testbed connects other objects to the component model. In this example, we connect four generic 'nodes' to the component model at the four given I/O locations. Data sent from these nodes to the component model traverse the given network, experiencing any network effects that are modeled, before exiting at another (or possibly the same) I/O point. Note that other testbed objects can exist that are not directly related to the component. We could connect other Extensible Components which encapsulate completely different modeling tools, or other physical nodes, or any other testbed object.

Since Extensible Components are first-class testbed objects, when the experiment is realized, the component is already correctly configured and operational, again reducing burden on the testbed user as there is no need to learn details of testbed operation in order to instantiate models. Importantly, a testbed user can specify an experiment with multiple models from one or more Extensible Components—all modeling different, and potentially complex, network systems which the testbed user can link together as needed.

Our WAN Extensible Component was used to generate multiple WAN models. To date, we have over 30 models associated with our Extensible Component, most of which were utilized in a recent DARPA program. The majority are models tailored to stress test and validate a particular system, or models designed to demo a feature of a system. A handful of these models capture Field Expert knowledge on military and commercial networks. We found capturing the knowledge of the operational Field Experts in the form of Extensible Component models greatly reduced the effort and time needed to get everyone within the DARPA program on the same page for testing, validation and demonstrations. The testbed users could focus on their systems and spent less time on understanding the details of the operational networks their systems would be deployed on and less time digging into the internals of the testbed.

While models are typically defined by the Field Experts, within the DARPA program we found it was also valuable to provide multiple dynamic controls directly to the testbed user. Our WAN Extensible Component allows users to tune very global and very specific link-level properties during an experiment, enabling testbed users to customize testing for stress testing and corner-case testing.

```
set link1 [$ns duplex-link
          $mynode [$myc entry router1]
          1000Mb 0.0ms DropTail]
```

Fig. 6. Connecting a component

Current Implementation Performance. As the Modeling Experts for wide area networks, it is our responsibility to describe the capabilities and limitations of our Extensible Component. As a link-based model of a WAN, the capabilities are well understood—the component enables modeling link properties with high-fidelity. The artifacts of these capabilities are also well understood, though are complex to express. For example, how bandwidth shaping is conducted on a link can greatly vary the effects on a flow. Token bucket approaches can cause unexpected burstiness where as dynamic queues based on bandwidth and delay result in less burstiness but can be more resource intensive (reducing the overall throughput available). We have quantified many of these capabilities and limitations for a multitude of our link properties.

Understanding the fundamental limitations of the underlying back-end is also critical. To this end, we measured the packet per second (PPS) performance of our Click-based [10] back-end under a variety of scenarios using 72 byte packets. All our experiments were conducted on a dual CPU, 12 Core Intel Xeon CPU E5-2650 v4 at 2.20 GHz with 64 GBs of RAM and two 40 Gigabit Intel XL710 interface cards. To achieve best performance, we deployed our system using DPDK [13] and FastClick [3].

First, we obtained an upper bound on performance—the maximum PPS Click could process. Utilizing a queueless Click configuration, we paired each interface and achieved 36 million PPS per NUMA node on our hardware (a total of 72 million PPS). This may not be a true upper bound, as our traffic generation nodes cannot generate additional traffic. This still represents a useful practical upper bound, as our link-based modeling approach requires queues to accurately emulate multiple link properties. Queues represent a significant performance penalty, as they act as a point where threads context switch and move to process each packet. Simply enabling basic queuing reduced our performance by 90% to 4 million PPS per NUMA node. Fortunately, adding all of our other link emulation options had a negligible effect on performance.

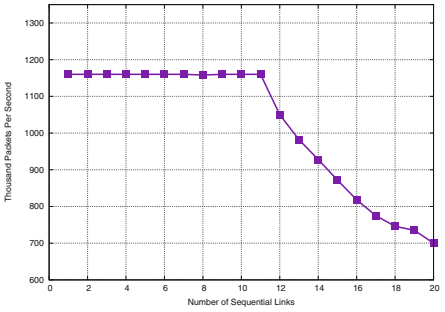

Fig. 7. Packets per second (PPS) throughput performance as we add additional sequential links in our WAN model.

It is also necessary to understand the performance of each emulated link. As each link can only be serviced by one thread at a time[1], there is a fundamental limit to how many packets a single link can process. Our experiments determined that an individual link can handle 1.2 million PPS. Further, placing multiple links in sequence has no effect on performance until twelve sequential links, after-which there is a steady decrease in performance. Figure 7 depicts this eventual decline in PPS as we add additional sequential links. This reduction in performance is most likely caused by thread contention and unexpected queuing delay throughout the sequential pipeline. As Modeling Experts, we must make it clear to any Field Experts that sequential links that do not have any branching points are best modeled as one single link.

The final performance question to ask is how many parallel links can be modeled before we experience artifacting. On our test setup, we have currently modeled up to 150 parallel links without experiencing loss of performance and are continuing to push this number until we begin to reach our limitations. It is also important to realize that placing bandwidth limitations on a link may reduce the effect of performance reductions. As a caveat, our results are directly related to the hardware tested on, and better hardware will have better performance.

4.2 Wireless Networks

Modeling wireless networks is vastly different than modeling wired networks. As part of a current DARPA program, we are in the process of developing a wireless emulation capability using Extensible Components. As modeling experts, we need to provide a modeling language for this component. The language is different than that of our WAN component as the notion of links does transfer to a broadcast medium. We are exploring a node-based, rather than edge-based, format that specifies the position, mobility, and transmission strength of our wireless speakers, amongst other properties.

As described previously, a tool is required to translate from our modeling language to a realization on the testbed. Further, a new back-end engine will be required to be able to conduct this realization. We are currently in construction of this back-end which focuses heavily on the network layer aspects of wireless, although many of the physical layer considerations are also being added in. Several critical constraints, such as bandwidth, delay and loss, are of course being developed. Additional capabilities are also being added including wireless collisions, channel access strategies and hidden terminals.

For modeling approaches of our Extensible Component, we are exploring a wide range of existing implementations that could be leveraged, including modifying Click [10], CORE [1], EMANE [14] or a combined integration (e.g. [2]). Until the back-end is fully realized, we will not be ready to develop our translation tool. However, understanding what can be modeled based on our language and our back-end heavily informs the capabilities of the overall system.

[1] This limit is important to prevent unexpected packet reordering or costly re-sequencing.

The final necessary building block for this new Extensible Component is the testbed interface. This portion is perhaps the simplest, as many of the relevant capabilities already exist in our library on DETER and our other existing components. To realize this new component, we need to make sure our new interface can accurately instantiate and realize the component on the testbed when the experiment becomes available. As stated previously, this is critical to ensure components remain first-class objects and are, therefore, easily usable for experimenters.

5 Related Work

Our work is related and motivated by a large body of research in network simulation and emulation. Many of these works offer tools which can be used in our Extensible Component Framework, and we expect to expand our model definitions beyond Click [10].

Network simulators, such as NS2/3 and OPNET [5] are widely used today. While tools like OPNET, share our goals in providing composable models to users, the extensibility of such tools is limited as these tools are largely focused on simulating an entire network, including using simulated traffic on a single piece of hardware, severely limiting how models built on these tools can interconnect. The ability to define a model across simulated *and* emulated components in these frameworks is limited. Likewise, network emulation tools such as Netem [8] and Dummynet [4] are focused on emulation, and lack the ability to define a model across both emulation and simulation. Later generations of NS and OPNET added "system-in-the-loop" capabilities, allowing traffic generated on external and real systems to be piped through the simulation. Despite system-in-the-loop capabilities, and a huge library of expert defined models, the ability to link OPNET models into a network of other models outside of the OPNET ecosystem is limited. Tools like NS3 and Dummynet offer enough flexibility to define composable components which can link into a range of other simulated or emulated components, but these tools still require a framework to capture complex models and incorporate field expertise.

Tools such as Emulab's NS Emulation (NSE) based on NS [6] and the U.S. Naval Research Laboratory's Common Open Research Emulator (CORE) [1] provide the ability to define a mix of simulation and emulation network components across a set of testbed resources. NSE's base-components are primitive and require additional framework to capture a complex model. NSE is no longer supported, but is an example of a toolkit which can easily benefit from our Extensible Components Framework. CORE [1] provides a 'network lab in a box' with a strong and usable API and an informative GUI. CORE still requires the user to have a good deal of domain knowledge on the networks being modeled as well as significant knowledge on the operation of CORE. CORE would be highly useful as a Extensible Component itself, which could be deployed through our framework to transfer domain knowledge from CORE experts to average experimenters.

Lastly, the Network Modeling and Simulation Environment (NEMSE) [9] works to provide unified access to a large number of emulation and simulation tools on the Emulab testbed platform [12], including CORE [1], OPNET [5] and Click [3,10]. While unified tool access is an important step towards sharing models, NEMSE lacks a framework to define complex and composable models and does not address treating models as first-class objects on its chosen testbed platform.

6 Conclusion

Complex and realistic scenarios are required to accurately evaluate and validate security and networking solutions. The knowledge required to generate faithful models continues to expand as the complexity of these systems and their environments grows. It is unrealistic to expect an experimenter will have or be able to obtain the necessary cross-domain expertise to generate these models and conduct scientifically valid experiments. Therefore, we must reduce the burden on these experimenters by capturing the knowledge of domain experts and disseminating this knowledge through first-class experimental objects to end-users.

In this paper, we described our Extensible Components Framework to perform knowledge capture and domain expertise transfer. A key realization incorporated into our framework is that knowledge should be captured and disseminated at three distinct levels—the modeling approach, the model creation and the model utilization. This separation allows knowledge to be captured by those who know how to describe their domain without necessarily knowing how to implement a model for their domain (Field Experts), as well as those who know how to accurately model a domain, but do not know the specifics of real-world networks and layouts (Modeling Experts). Further, all of this captured knowledge can be made available and usable to an experimenter without needing to obtain that domain knowledge. By treating Extensible Components as first-class testbed objects, experimenters can easily deploy and use complex models and scenarios without having to gain additional knowledge on the components implementation or configuration. It is our belief that this framework will promote rigorous experimentation and validation of security and networking systems by promoting the creation and sharing of accurate models.

References

1. Ahrenholz, J.: Comparison of CORE network emulation platforms. In: Military Communications Conference, 2010 - MILCOM 2010, pp. 166–171, October 2010. https://doi.org/10.1109/MILCOM.2010.5680218
2. Ahrenholz, J., Goff, T., Adamson, B.: Integration of the core and emane network emulators. In: 2011 - MILCOM 2011 Military Communications Conference, pp. 1870–1875, November 2011. https://doi.org/10.1109/MILCOM.2011.6127585
3. Barbette, T., Soldani, C., Mathy, L.: Fast userspace packet processing. In: Proceedings of the Eleventh ACM/IEEE Symposium on Architectures for Networking and Communications Systems, ANCS 2015, pp. 5–16. IEEE Computer Society, Washington (2015). http://dl.acm.org/citation.cfm?id=2772722.2772727

4. Carbone, M., Rizzo, L.: Dummynet revisited. ACM SIGCOMM Comput. Commun. Rev. **40**(2), 12–20 (2010). https://doi.org/10.1145/1764873.1764876
5. Chang, X.: Network simulations with OPNET. In: Proceedings of the 31st Conference on Winter Simulation: Simulation–A Bridge to the Future, WSC 1999, vol. 1, pp. 307–314. ACM, New York (1999). https://doi.org/10.1145/324138.324232
6. Fall, K.: Network emulation in the Vint/NS simulator. In: Proceedings of the fourth IEEE Symposium on Computers and Communications, pp. 244–250 (1999)
7. Floyd, S., Kohler, E.: Internet research needs better models. ACM SIGCOMM Comput. Commun. Rev. **33**(1), 29–34 (2003). https://doi.org/10.1145/774763.774767
8. Hemminger, S.: Network Emulation with NetEm. In: Linux Conf AU (2005). http://www.linux.org.au/conf/2005/abstract2e37.html?id=163
9. Hench, D.L.: Complex network modeling with an Emulab HPC. In: Proceedings of the 16th Annual IEEE High Performance Extreme Computing Conference (HPEC 2012), September 2012
10. Kohler, E., Morris, R., Chen, B., Jannotti, J., Kaashoek, M.F.: The click modular router. ACM Trans. Comput. Syst. **18**(3), 263–297 (2000). https://doi.org/10.1145/354871.354874
11. Mirkovic, J., Benzel, T.V., Faber, T., Braden, R., Wroclawski, J.T., Schwab, S.: The DETER project: advancing the science of cyber security experimentation and test. In: 2010 IEEE International Conference on Technologies for Homeland Security (HST), pp. 1–7, November 2010. https://doi.org/10.1109/THS.2010.5655108
12. White, B., et al.: An integrated experimental environment for distributed systems and networks. In: Proceedings of the Operating System Design and Implementation, pp. 255–270 (2002)
13. DPDK: Data Plane Development Kit. https://dpdk.org/
14. Extendable Mobile Ad-hoc Network Emulator (EMANE). https://www.nrl.navy.mil/itd/ncs/products/emane