SmartTags: Bridging Applications and Network for Proactive Performance Management

Ali Munir, Seyed Hossein Mortazavi, Mahmoud Mohamed Bahnasy,
Amir Baniamerian, Shimiao Wang[2] , Shichao Guan, Yashar Ganjali[1,2]
[1]Huawei Technologies Canada Co. Ltd., [2]University of Toronto
ali.munir@huawei.com

ABSTRACT

Sudden changes in the applications and events in the network are often related. Many of the datacenter applications go through sudden state changes (such as a query-response in MemCached application) that may result in an event in the network (such as utilization, packet drops, etc.). Existing works do not fully leverage the relationship between application state changes and network events and as a result provide limited performance improvements. An ideal application and network management system should be able to automatically identify the sources of sudden changes in the application, host or network and relate these changes to network events to enable proactive network management. In this work, we propose SmartTags, a system that automatically learns which of the application state changes (Tags) are related to the network events, and uses this information for proactive network management. At a high level, smartTags is orthogonal to current NAI approaches. It provides a systemic way for application developers and network designers to automatically learn the relationship between application behavior and network events. Through very simple small scale real testbed based experiments, we demonstrate that smartTags can improve the training time of a distributed machine learning application by 27% while minimizing loss to zero. Similarly, it can improve the query completion time of MemCached by 32% while achieving near zero loss. We envision much more gains in large scale distributed systems.

CCS CONCEPTS

• **Networks** → **Network protocol design**.

KEYWORDS

SmartTags, Datacenter Networks, NAI

ACM Reference Format:
Ali Munir, Seyed Hossein Mortazavi, Mahmoud Mohamed Bahnasy,, Amir Baniamerian, Shimiao Wang[2] , Shichao Guan, Yashar Ganjali[1,2]. 2022. SmartTags: Bridging Applications and Network for Proactive Performance Management. In *ACM SIGCOMM 2022 Workshop on Network-Application Integration (NAI '22), August 22, 2022, Amsterdam, Netherlands.* ACM, New York, NY, USA, 7 pages. https://doi.org/10.1145/3538401.3546601

NAI '22, August 22, 2022, Amsterdam, Netherlands

ACM ISBN 978-1-4503-9395-9/22/08…$15.00
https://doi.org/10.1145/3538401.3546601

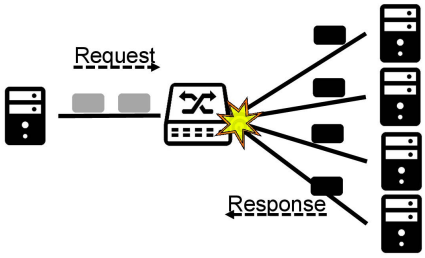


Figure 1: Causal Relation between Application Changes and Network Events. A request from the application can create a simultaneous response from multiple servers resulting in packet drops in the network.

1 INTRODUCTION

Sudden changes in the applications and events in the network are often related. Datacenters have many distributed components working together to run applications and network. These components often go through many sudden state changes (such as an application switching from data processing to sending data over the network) that result in various network events (such as congestion). For example, when a worker in the distributed machine learning (DML) [8] or a database application [6] finishes processing data and switches to communication phase to transfer the processed data over the network, it may start many flows simultaneously sending data to the same destination. This sudden change in the large amount of data may create congestion in the network and hence may result in packet drops (Figure 1).

Many existing mechanisms have leveraged the information from the applications for proactive network management to improve the application and network performance. For example, an epoch start time aware traffic management solution is proposed to proactively handle network congestion caused by DML applications [7]. Similarly, many scheduling mechanisms leverage flow size information to improve the application performance [9, 13].

To the best of our knowledge, existing works do not fully leverage the relationship between application state (changes) and network events and as a result the proposed solutions are limited to specific use cases only. This is mainly due to two reasons.

First, the fundamental limitation of the prior art is the lack of ability to "automatically" associate application state changes with the network events. For example, applications can not inform the network of the state changes, and as a result the network is unaware of when, how much, or, to how many data will an application send. Similarly, network is a black box for the application, and as a result applications are unaware of how their communication patterns affect the network state (e.g., if it creates congestion or not.). Hence,

any new solution needs to learn what application state changes impact the network and how.

Second, existing solutions have limited ability to detect and adjust to the sudden changes. Sudden changes in application or endhost require time to prepare and adjust network behavior. As a result, many of the solutions are reactive in nature that detect application changes when something has already happened in the network. For example, congestion control depends on the network events (loss in TCP [3] and delay in BBR [1]) to detect if there is congestion in the network and then do preventive measures to avoid congestion.

An ideal application and network management system should be able to automatically identify the sources of sudden changes in the application, host or network and relate these changes to network events to enable proactive network management. Recently, Socker [4] and socketIntent [10] have been designed to share information between application and network. However, these frameworks have very limited functionality and are not capable of exchanging state change information with the network. Even if these interfaces enable information exchange, it is hard to identify which of the application information is related to network events.

In this work, we propose SmartTags, a distributed system that automatically learns which state changes (represented as Tags) are related to the network-wide events (Events), and uses this information for proactive network management. At a high level, SmartTags is orthogonal to current NAI approaches. It provides a systemic way for application developers and network designers to automatically learn the relationship between application behavior and network events and identify which state changes can indicate network state.

For SmartTags, Tags are informative, lightweight, and predictive signals from the "application or endhost" that represent the "state" or "changes in the state" of an "application, host, or network". *In this work, we assume that Tags are generated by the application and are provided as an input to SmartTags.* For example, at the application level, for a MemCached application "MultiGet" is a common method to collect response for multiple queries simultaneously. Whenever the application wants to issue this function, it can trigger a signal which acts as a Tag for the SmartTags system. *SmartTags then learns if this Tag is responsible for any event in the network.* Whereas, Events are the changes in the network such as bursts, packet drop, utilization etc.

SmartTags is a scalable and generic system that operates in two phases. First, it collects and learns the relationship between Tags and Events in an offline manner. For this purpose, we propose a SmartTags protocol that defines a generic definition (format) of the Tags (or Events), and a compact and expressive SmartTags signature that represents the relation between Tags and network events. Second, it proactively adjusts the network configurations in real-time based on the Tags. For this purpose, we propose a distributed architecture for Tags and Events collection and for proactive network management in real-time.

Using two usecases, DML and MemCached, we demonstrate two key features of the SmartTags system: First, it can very reliably identify which of the sudden changes in application state (Tags) are most responsible for the network-wide events. Second, it is predictive and proactive. SmartTags has the ability to predict network events several time units (RTT) earlier even before they happen.

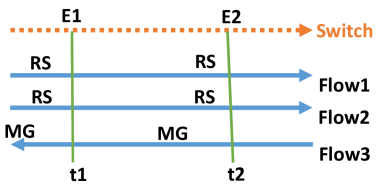


Figure 2: Predictiveness - Application state changes (Multiget query (MG), and response (RS)) always happen before the network events (E1 and E2) such as packet loss.

Therefore, it opens up possibilities to design solutions that can prevent undesired events (such as packet drops) from even happening in the network.

Through very simple small scale real testbed based experiments, we demonstrate that SmartTags can improve the training time of DML by 27% while minimizing loss to zero. Similarly, it can improve the query completion time of MemCached by 32% while achieving near zero loss. We envision much more gains in large scale distributed systems.

2 MOTIVATION

Many applications, such as distributed machine learning (DML) [8] or distributed databases, that are repetitive in nature exhibit a strong correlation between the application state changes and the network events.

2.1 Application state change impacts network

In a distributed MemCached application, first a client sends a request (e.g., "MultiGet") to multiple servers. Next, the servers respond to the query which can result in packet drops at the switch depending on how many servers respond to the queries simultaneously. This request response process repeats for every single request. For such repetitive applications, we observe the following properties.

Prediction of network events. The application state changes can hint about the network events ahead of time. For example, in Figure 2, we can see that the application state change in the form of Multiget request (MG) or the server response (RS) always happens before the packet drop events (E1 and E2) in the network.

Accurate Relation. The application state changes can indicate with a very high accuracy if an event is going to happen in the network. To demonstrate this, we run a simple MemCached test case with 200 MG requests to 5 different servers. We assume that the data is stationary for the use-case. We compare the time to packet drop after sending an MG request with a random sampling based approach. We can see that there is a very high correlation between the MG and packet drop event in the network. As a result, in Figure 3, one can tell with a very high accuracy that there is going to be a packet drop event within 200μs. On the other hand, random sampling based approach is unable to give any insights about the packet drops in the network. In this experiment, we consider a random sampling approach where we randomly sample the switch to observe any packet drops. In this case as the time at which the packet drop happen, with respect to the application state, is not fixed. Hence it is very hard to relate packet drops in the network with the application state.

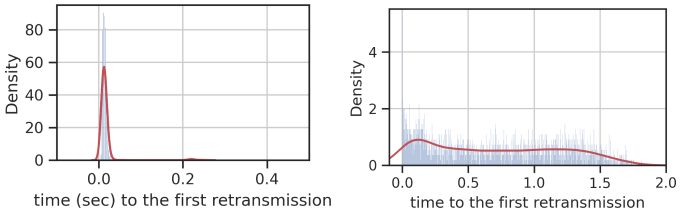


Figure 3: Application state changes can hint with very high accuracy if an event is going to happen within a certain duration (left), compared to existing random sampling based approach (right).

2.2 Why SmartTags?

For a true application network integration, it is essential to learn the relationship between applications and network and how they impact each other's performance. SmartTags system aims to fill this gap by automating the process of relating the application state changes (represented as Tags) with the network events for proactive network management. Note that the Tag is not limited to application state changes only, it can be the socket level connection state as well.

For example, the Tag MG in MemCached example can be obtained when the query is generated and SmartTags can learn its relation with the events (such as burst) in the network. Later this Tag can be used to proactively prepare the network to handle incoming burst.

SmartTags enables many potential applications: it can be used for building smart endhosts that do Tags aware flow control / scheduling (similar to [9]), smart switches that do Tags aware congestion signaling, routing, load balancing etc., (similar to [7]), and smart applications that do Tags aware dynamic application logic, DAG scheduling.

3 SMARTTAGS

In this section, we share our vision of how a system like SmartTags may be designed.

3.1 Tags and Events

Tags are lightweight signals generated at the application, endhost or network that represent the state or changes in the state of an application, a host, or network (like in any of the existing works). Tags, in SmartTags, are generic and at the same time application specific. SmartTags does not require any global agreement between the Tags from different applications as compared to existing systems. Moreover, Tags do not contain any information about how it is related to other Tags or Events. Tags also do not require/contain any detailed information about application content itself (to maximize data privacy).

Events are the signals from the network that represent different statistics about the network state, such as delay, queue build up or packet drops etc.

Application level Tag examples are (not limited to) application type (stream/bursty flow/map/reduce/...), application state (response/request, number of repetition, Start/End time, direction) etc. Host level Tag examples are App usage statistics (such as CPU cycles used, Disk I/O, Memory I/O) or Socket level parameters (such as Inter flow gap, start/end time, flow size, congestion window size change, message buffer size, priority etc.) Similarly, examples of network events are burst, packet drops, queue size, load etc.

3.2 SmartTags Protocol

SmartTags protocol defines what and how the Tags and Events information needs to be collected. To make SmartTags generic (i.e., represent a wide variety of applications) and have a uniform representation format of Tags and Events, we propose a specific format for collecting the Tags and Events in the network. Tags and Events are collected in the following format – *time, localeID, Tag (or Event), value*. The *time* represents the instant at which the Tag was generated (or the Event was reported), *localeID* represents the origin of the information, it could be an Application ID, the Job ID, or a specific flow ID (+ 5-tuple), or the host/switch ID, or a combination of these. The *Tag* can be a generic identifier that represents the state change.

Note: It is the precise information *Time, localeID, Tag, Value* along with the *Tag* that helps make it smart. In SmartTags, *Time (when), Location (where), Tag (or Event) (what)* are the must-haves information. *Time* and *LocaleID* identifier are the key elements that help in identifying the related state changes and events, and make *Tags* smarter. Optional info such as *extent (value)* can further help enhance/improve the system and make *Tags* smarter. For example, In MemCached, for a Multiget request if we don't know the query size, we can still use SmartTags system, but additional info helps in building better solutions.

3.3 SmartTags Signature

The SmartTags signature represents the stochastic relation between the Tags and Events, and it accurately captures when and where an Event is going to happen if a specific Tag is seen. The key requirements in designing a signature are that it should be compact and provide an expressive representation of Tags and related Events. Moreover, the signature should be usable in fast and online manner in data plane at line rate.

To generate a signature, the SmartTags requires the Tags, Events, and the network graph showing relationship between components (such as switches, hosts, application) as an input. The output is the signature that gives the probability of an Event happening at a location within certain time (td), given a Tag i.e., *Prob(td, localID, Event | Tag)*.

For signature generation, First, SmartTags groups the Events and Tags based on the location (and relationship) information, and then it segregates them based on the Events and Tags happening within a certain time window. Next, for each cluster it estimates the probabilities of the Tags and their relationship. Note that this is just one potential way to generate signatures. SmartTags can use any of the existing machine learning or statistical approaches [14, 15] to find these signatures. The goal of this work is to present a proof-of-concept and we leave optimizing the design of an efficient signature generation mechanism to future work.

SmartTags signature can capture the relationship between Tags and Events even if Tags from only a subset of the applications are

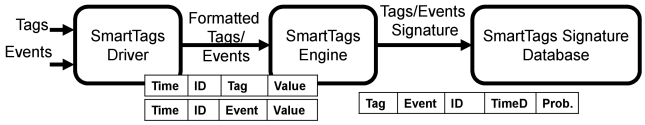


Figure 4: SmartTags uses the Tags and Events to generate a signature with the information of when and where an Event is going to happen after seeing a Tag. This is then used for proactive network management.This is an offline process that happens at the start (and updated periodically) while the application is running.

available. The probability captures the effect that if an Event is happening due to the Tag or the background applications.

3.4 SmartTags Design & Implementation

SmartTags is an end-to-end system for proactive network management based on Tags that works in two phases:

Identification Phase. In this phase, the SmartTags system collects and correlates Tags (and Events) from the network and applications, as shown in the Figure 4. The input to this phase are the Tags and Events and output is the SmartTags signature, that represents the relationship between the Tags and Events, for the network controller.

Action Phase. In this phase, the SmartTags system monitors Tags and takes real-time actions in the data path to achieve desired performance objective. The smart network controller uses the signature to create rules and actions for the datapath that are proactively installed by the control plane. For example, a SmartTags enabled network controller can proactively update the forwarding tables at the switches to reroute flows based on the SmartTags signature and avoid congestion in the network.

SmartTags is an iterative system that keeps on updating the signature to capture accurate correlation (C) between Tags and Events based on the current network conditions. The identification phase runs in an offline phase, that is repeated periodically to update the learned relationship based on changing network behaviors. Action phase on the other hand runs in real-time and leverages the Tags and the learnt signatures for proactive network management.

Two phase design makes SmartTags suitable for applications that are iterative and repetitive in nature. During the initial runs, SmartTags can learn the application behavior and its impact on the network, without affecting the application performance, and later it can proactively manage application and network behavior. Moreover, this design eliminates the need for the SmartTags engine to operate at the RTT granularity, thus improving its ability to be used in real-time.

To realize SmartTags in a large scale distributed datacenters, this work implements following components, Figure 5.

3.4.1 SmartTags Drivers (S.D.). S.D. are the muscles that collect and forward Tags to the SmartTags engine for generating signatures or the smart network controller to take real-time actions to achieve desired performance objective. S.D. is a distributed element that runs on hosts and/or switches. Switches may use existing controllers/monitoring systems to report Events to the S.D.

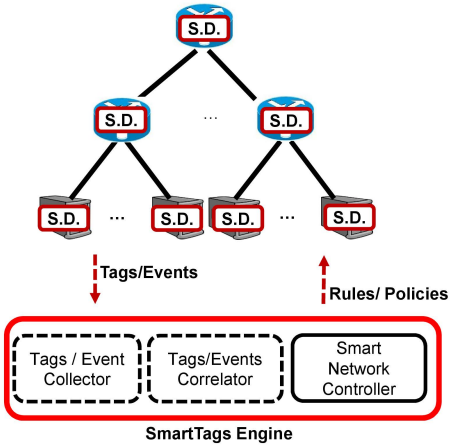


Figure 5: SmartTags Distributed Components (in red).

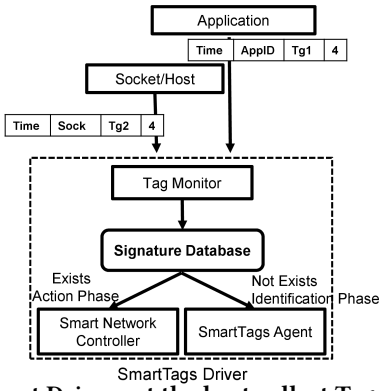


Figure 6: Smart Drivers at the host collect Tags from applications/hosts and share it with the SmartTags engine and network controller.

SmartTags Driver at the host: Figure 6 shows the key elements of the S.D. on a host machine. The S.D. is implemented as a driver that interacts with the application, host and the network controller. It collects Tags from the application or the host using the process called "Tag Monitor" and checks in the local database if a signature for this Tag exists locally. If the signature exists, the Tag is forwarded to the smart network controller, which takes certain actions based on the signature (action phase). If the signature does not exist, the Tag is forwarded to the SmartTags agent (identification phase). The SmartTags agent supports correlation of the local Tags and local Events and forwards the Tags to the SmartTags engine for correlation with the global network events. We implement S.D. drivers as python module at the endhost.

SmartTags Driver in the network: The S.D. interacts with the switch's flow and event monitors. It collects Tags and events from these entities using 'Tags Monitor' and checks in the local database if a signature for this Tag exists locally. If the signature does not exist then the Tag is forwarded to the S.E. via SmartTag agent. If the signature exists, the Tag is forwarded to the switch controller, which takes certain actions based on the signature, such as reroute packets or change queue thresholds.

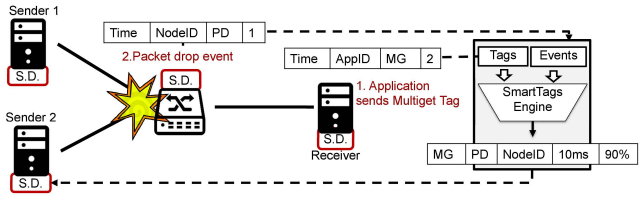


Figure 7: SmartTags collects 1) Tag, and 2) Event. Next, it learns the relationship between Tags and Events and generates signatures for the network controllers.

3.4.2 SmartTags Engine (S.E.). S.E. is the brain that enables identifying the relationship between Tags and Events. It is a centralized (logically) entity that collects Tags (and Events) from the SmartTags agents at the hosts and switches during the identification phase. It correlates Tags (and Events) and generates signatures to help define rules and policies. The signatures are added in the S.D. database for use by the smart Network controllers, during the actioon phase.

3.4.3 Smart Network Controller. It enables proactive solutions to address various network challenges (such as taking early actions to avoid burst of data or packet drops) using SmartTags signatures. It takes the SmartTags signature as an input and produces rules and action policies for proactive network management in the format of rule, action, expiration. The expiration tells the duration for which a rule should be active in the network. If a Tag is seen, the smart network controller generates and installs a rule action and sets it to delete after the "expiration" time.

3.4.4 Implementation. Implementing SmartTags requires changes in both the hosts and the network switches. We envision SmartTags to run as a service that can work with the applications and network controllers to achieve its objectives. For example, the SmartTags driver at the host can interact with the applications through socket interfaces such as Socker [4], or it can be a dedicated service with which the application can register and share its state. Another option might be adding placeholders in the application code that can generate Tags when executed. More details on this can be found in our related work [11]. In this work, we adopt this approach for design simplicity and ease of developing a proof of concept.

Similarly, the controller implementation is very usecase specific and can be done using existing SDN controllers or other technologies. In this work, we implement a simple controller at the endhost that reroutes traffic (using linuxipTables) or adds delay (using native linux traffic control utility like tc) to the packets upon observing specific Tags.

This distributed design makes SmartTags scalable as it avoids contacting the centralized S.E. every time a Tag is observed. S.D. can generate a rule action locally once it sees a Tag. SmartTag drivers and engine communicate over any message communication protocol (such as ZMQ).

Lastly, the SmartTags driver at the switches can be implemented through programmable switches and software defined networking. We discuss one such design in the following work [5]. Moreover, many switches provide telemetry services that can be configured to collect network events, similar to NetHint [2].

4 CASE STUDIES

We demonstrate SmartTags benefits using two applications representing machine learning and database systems.

4.1 SmartTags for MemCached

For a MemCached system, as shown in Figure 7, during the identification phase: First, the receiver sends a Multiget (MG) request to two senders at time t1 – application generates a Tag (MG) showing the request. Second, the sender response (RS) results in packet drops at time t2 – which is logged as an Event at the switch. Lastly, the SmartTag engine collects and correlates Tags and Event to generate a signature for the smart network controller. The controller then installs the delay rules in the S.D. to mitigate packet drops in the future.

During the action phase: First, the receiver sends a request to two senders at t1 and application generates a Tag showing the request. Second, the SmartTags driver at the host sees the Tag and adds random delay to the flows going out of the sender. As a result packets from different senders are desynchronized and hence the network avoids packet drop.

In our testbed, we modify the MemCached application to generate a Tag whenever, it issues an MG request or generates RS. Furthermore, we add a delay to the response using the linux utility 'tc' to proactively manage congestion based on the Tag.

4.2 SmartTags for DML

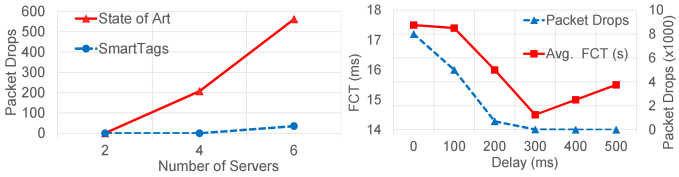
For a DML application, we consider a synchronized parameter server [8] architecture. Similar to MemCached, during the identification phase: First, the parameter server (PS) runs an "rpc_sync" function at t1 which pulls the updated parameters from the worker nodes, application generates a Tag showing the request. Next, the worker nodes send parameters to the PS that may result in packet drops at t2 – which is logged as an Event at the switch. Lastly, the SmartTags engine generates a signature. For our study we consider rerouting the flows of the worker nodes, when a Tag is seen for a flow. For this we configure two paths between the parameter server and each worker. Upon seeing the "rpc_sync", the worker nodes select one of the two paths for the outgoing flows.

In our testbed, we modify the pytorch application code to generate a Tag whenever an 'rpc_sync' command is executed. Furthermore, we add a re-route rule using the linux utility 'iptables' to proactively manage congestion based on the 'rpc_sync' Tag.

4.3 Technical Benefits Demo

We run above two applications. First, we run a simple MemCached test case with 200 MultiGet (MG) requests to 5 different servers. Second, we test with a parameter server based distributed machine learning application, with one server and 6 workers using ResNet-18 model.

SmartTags is Generic. SmartTags captures any Tags from the application and Events. For example, Tags from the DML Application $t1, DML, S2W, 4$ represent flow of data from server to 4 workers at $t1$. Tags from the MemCached $t2, MC, MG, 80$ represent the multiget request from 80 values at $t2$. And, Events (burst) from the switch $t3, SW1, PD, 10$ shows 10 packet drops at switch1 at $t3$.



(a) Packet Drops DML **(b) FCT / Drops MemCached**

Figure 8: SmartTags improves the performance by proactively managing congestion in the network. b) impact of different actions (varying delay) on application performance. SmartTags can help predict the desired amount of delay. For example, in this experiment delay of 300ms can mitgate packet drops and improve FCT. Adding any more delay degrades the application performance.

SmartTags captures Relation. SmartTags can capture relation between Tags and Events. For example, for the DML application we can see that as the number of servers (a unique Tag for each) increase we observe an increase in the packet drops, Figure 8a. Similarly, for the Memcached Multiget of different sizes show a relation with the loss rate. Thus, SmartTags can tell how the Tags are related to Events in the network. This is further highlighted in Figure 3, as discussed in § 2.1.

SmartTags is Predictive. As demonstrated in Figure 3 (§ 2.1), SmartTags can predict an event much earlier as compared to in-network solutions that are unaware of application state changes. For example, in MemCached, a burst can be predicted 200msec earlier, which is significantly larger (1000x) than the latencies of current datacenter networks. Similarly, in DML depending on the model these gains could be even higher. This allows sufficient time for the controller to reconfigure network to handle undesired events.

SmartTags is Proactive. SmartTags enables proactive network management. To test this we installed two different solutions. First, for the DML application, we observe that rerouting (physically distinct paths) flows can mitigate packet drops caused by the bursts and improve training time (Figure 8a). Similarly, for the Memcached application, flows are delayed if a Tag is seen. We can observe from the Figure 8b that as we increase the amount of delay, the number or packet drops reduce significantly. Zero delay here represents the state-of-art.

5 DISCUSSION AND CONCLUSION

5.1 Differences from Prior Art

SmartTags has following key differences from the prior art. First, existing solutions use a predefined set of Tags that provide limited use in specific cases only [9]. Second, some works are reactive by design, i.e., they react based on the network events e.g., multiplicative decrease upon loss in TCP. Third, some works find relation between Tags but do not consider effect on the network, e.g., flow size estimation [13]. Fourth, they find relation between Events, do not consider effect of the Tags on these Events [15]. Lastly, some works use the Tags from the applications for proactive network management [7].

SmartTags extends the capabilities of these systems as follows: First, Tags are generic and at the same time application specific. SmartTags does not require any global agreement between the Tags from different applications as compared to existing systems. Second, it automatically identifies the chance (probability) and degree of relationship of a Tag with the network event. It is not limited to a fixed set of Tags or Events. It creates a signature that can be used to define any rule/action on the fly. Third, SmartTags is proactive as it provides a prediction on network state (couple of RTT ahead) that gives the network enough room to define new rules and measures to avoid congestion. Lastly, SmartTags finds the relation between Tags and Events. Although it is capable of doing between only Tags or only Events too.

5.2 Potential Future Directions

Signalling between Applications and SmartTags. A key design question for smartTags realization is how to share the information between applications and the SmartTags system. In this work, we follow a simplistic approach of modifying the application code to add potential signals. However, such a solution is impractical to implement. A potential direction could be to automate the placeholder selection process, as in EarlyBird [11] or using distributed application tracing, as in Dapper [12].

We demonstrate (EarlyBird [11]) a potential way to automate the placeholder selection in the application code. EarlyBird helps application developers identify what can be the potential locations to add Tags in their code.

SmartTags Prediction Accuracy. Another key design question for SmartTags is how accurately can SmartTags capture the causal relationships between Tags and Events. In this paper, we demonstrate that in isolated environment where only a single application is running, this could be done very efficiently and accurately. However, larger datacenters have complex environment where multiple applications co-exist. In such settings, one single event may be the result of multiple application state changes or the combination of the behavior of different applications. We would like to highlight that SmartTags systems captures the probabilistic relation between Tags and Events, which looks at the long term behavior of the application and network. Therefore, it can easily detect if an event is persistently being caused due to the changes in application state. We leave detailed evaluation of this as a future work.

We demonstrate (in [11]) that the SmartTags system can identify the true causal relationship between Tags and local Events at the host with a very high accuracy. We also highlight the imapct of td on the correlation probability for the SmartTags signature. Smaller td can lead to false negatives, failing to capture the relationship while larger td can lead to false positives. We leave detailed evaluation of this as a future work.

SmartTags opens doors for much richer coordination between the applications and network. To the best of our knowledge, none of the previous works consider Tags from the application state changes and their effect on network events. SmartTags is defining a way for intelligence sharing between the applications and network. This opens up many possibilities for proactive solutions. In this work, we propose a general system architecture to support our vision.

REFERENCES

[1] Neal Cardwell, Yuchung Cheng, C Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. 2017. BBR: congestion-based congestion control. *Comm. ACM 2017* 60, 2 (2017), 58–66.

[2] Jingrong Chen, Hong Zhang, Wei Zhang, Liang Luo, Jeffrey Chase, Ion Stoica, and Danyang Zhuo. 2022. NetHint: White-Box Networking for Multi-Tenant Data Centers. In *19th USENIX Symposium on Networked Systems Design and Implementation (NSDI 22)*. 1327–1343.

[3] Sally Floyd and Van Jacobson. 1993. Random early detection gateways for congestion avoidance. *IEEE/ACM ToN 1993* 1, 4 (1993), 397–413.

[4] Dong Guo, Shuhe Wang, and Y Richard Yang. 2021. Socker: Network-application Co-programming with Socket Tracing. In *ACM SIGCOMM NAI 2021*. 14–19.

[5] Dong Guo, Shuhe Wang, and Y Richard Yang. 2022. NCE: An ECN Dual Mechanism to Mitigate Micro-bursts. In *ACM SIGCOMM NAI 2022*. 14–19.

[6] Jithin Jose, Hari Subramoni, Miao Luo, Minjia Zhang, Jian Huang, Md Wasi-ur Rahman, Nusrat S Islam, Xiangyong Ouyang, Hao Wang, Sayantan Sur, et al. 2011. Memcached design on high performance rdma capable interconnects. In *ICPP 2011*. IEEE, 743–752.

[7] Minkoo Kang, Gyeongsik Yang, Yeonho Yoo, and Chuck Yoo. 2020. Proactive congestion avoidance for distributed deep learning. *Sensors 2020* 21, 1 (2020), 174.

[8] Mu Li, Li Zhou, Zichao Yang, Aaron Li, Fei Xia, David G Andersen, and Alexander Smola. 2013. Parameter server for distributed machine learning. In *NIPS 2013*, Vol. 6. 2.

[9] Ali Munir, Ghufran Baig, Syed M Irteza, Ihsan A Qazi, Alex X Liu, and Fahad R Dogar. 2014. Friends, not foes: synthesizing existing transport strategies for data center networks. In *ACM SIGCOMM 2014*. 491–502.

[10] Philipp S Schmidt, Theresa Enghardt, Ramin Khalili, and Anja Feldmann. 2013. Socket intents: Leveraging application awareness for multi-access connectivity. In *CoNext 2013*. 295–300.

[11] Mahmoud Mohamed Bahnsay et. al. Seyed Hossein Mortazavi, Ali Munir. 2022. EarlyBird: Automating Application Signalling for Network Application Integration. In *ACM SIGCOMM NAI 2022*.

[12] Benjamin H Sigelman, Luiz Andre Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, and Chandan Shanbhag. 2010. Dapper, a large-scale distributed systems tracing infrastructure. (2010).

[13] Vojislav Đukić, Sangeetha Abdu Jyothi, Bojan Karlaš, Muhsen Owaida, Ce Zhang, and Ankit Singla. 2019. Is advance knowledge of flow sizes a plausible assumption?. In *USENIX NSDI 2019)*. 565–580.

[14] Ting Wang, Mudhakar Srivatsa, Dakshi Agrawal, and Ling Liu. 2009. Learning, indexing, and diagnosing network faults. In *ACM SIGKDD 2009*. 857–866.

[15] Ting Wang, Mudhakar Srivatsa, Dakshi Agrawal, and Ling Liu. 2010. Spatio-temporal patterns in network events. In *CoNEXT 2010*. 1–12.